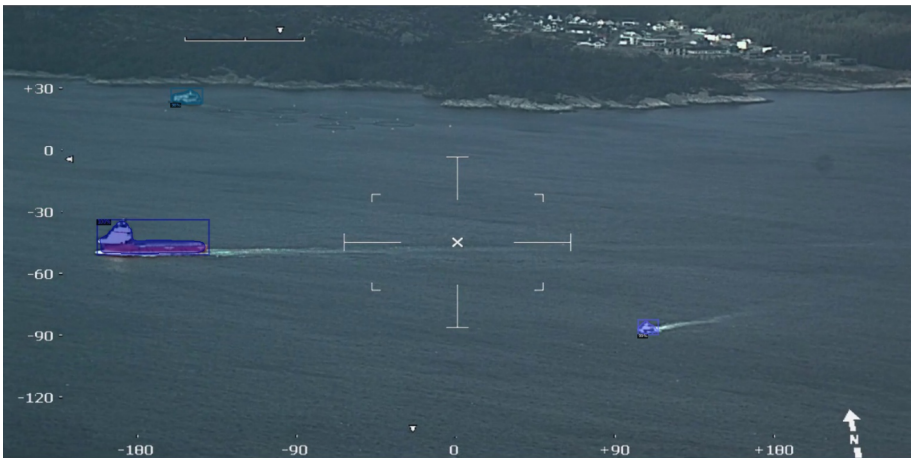
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Paul Vik

Detection of Naval Vessels using Deep Learning and Aerial Images

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
June 2020



NTNU
Norwegian University of Science and Technology

Paul Vik

Detection of Naval Vessels using Deep Learning and Aerial Images

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

The scope of this thesis is to create a deep learning pipeline with the aim of detecting naval vessels from aerial imagery. The pipeline involves creating a custom dataset by gathering images from NSMs Seahunter system and annotating them. The generated dataset is subsequently used to train deep convolutional neural networks. Instance segmentation architectures were utilized for this purpose. However, their performance was only evaluated by bounding box average precision scores rather than segmentation scores. This decision was made such that the individual performances would be comparable with previous work within object detection.

The process of annotating images with masks is time-consuming. This thesis addresses this by proposing *initial automatic annotation* to accelerate this process. This method utilizes a COCO pre-trained model to generate annotation proposals to reduce the number of manual annotations. The process reduced the manual annotation load with 24.16 % on the additional images that supplemented the existing dataset. In total, a training set of 3,941 images and 4,693 objects were annotated with masks.

Two instance segmentation architectures, Mask-RCNN and Cascade-RCNN, were selected for this thesis. The choice was made primarily based on their widely proven accuracy and accessibility within the Detectron2 library. Two main training strategies were utilized, transfer learning and training from scratch, i.e. fully training the network. A hybrid approach of the two was proposed, which was to fully train the network initialized with ImageNet pre-trained weights. The hybrid approach surprisingly outperforms both the transfer learned and the default, fully trained models. Mask-RCNN with Resnet50 as backbone trained with the hybrid approach gains a bounding box AP score of 44.5 on a test set of 1,516 images. The results indicate that it is viable to fully train deep convolutional networks on smaller datasets down to 4,000 training images. They also indicate that fully trained networks are an effective alternative to the transfer learning paradigm within computer vision.

Sammendrag

Omfanget av denne oppgaven er å lage en strømlinjet dyp lærings prosess med mål om å oppdage marinefartøyer fra luftfoto. Prosessen innebærer å lage et tilpasset datasett ved å samle bilder fra NSMs Seahunter-system og markere fartøy i dem. Det genererte datasettet blir deretter brukt til å trene dype nevrale nettverk. Instans-segmenteringsarkitekturer (eng: instance segmentation architectures) ble brukt til dette formålet. Imidlertid ble deres ytelse bare evaluert ved gjennomsnittspoeng for omkretsbokser i stedet for segmenteringsmasker. Denne avgjørelsen ble tatt slik at de individuelle modellene ville være sammenlignbare med tidligere arbeid innen objektdeteksjon.

Prosessen med å markere fartøy i bilder med masker er tidkrevende. Denne oppgaven tar for seg dette ved å foreslå initiell automatisk markering for å akselerere denne prosessen. Denne metoden bruker en COCO-forhåndsopplært modell for å generere forslag til markeringer for å redusere antall manuelle markering. Prosessen reduserte den manuelle markeringsbelastningen med 24,16 % på de ekstra bildene som kompletterte det eksisterende datasettet. Totalt ble et treningssett med 3,941 bilder og 4,693 objekter merket med masker.

To instans-segmenteringsarkitekturer, Mask-RCNN og Cascade-RCNN, ble valgt for denne oppgaven. Valget ble først og fremst basert på deres nøyaktighet og tilgjengelighet i Detectron2-biblioteket. To hovedtreningsstrategier ble benyttet, overføring av læring og trening fra bunnen av, dvs. full trening av nettverket. En hybrid tilnærming av de to ble foreslått, som var å trene nettverket med ImageNet ferdig trente initialiserte vekter. Hybridtilnærmingen overrasker med bedre ytelse enn både overførte lærte og fullt trente modeller. Mask-RCNN med Resnet50 som ryggrad og trent med hybridtilnærmingen får en avgrensningsboks-AP-poengsum på $44,5$ på et testsett med 1 516 bilder. Resultatene indikerer at det er mulig å trene dype nevrale nettverk på mindre datasett ned til 4000 treningsbilder. De indikerer også at fullt trente nettverk er et effektivt alternativ til transfer learning-paradigmet innen datasyn.

Preface

This report was written on the behest of Norwegian Special Mission (NSM) as part of the master thesis TTK4900 at the Norwegian University of Science and Technology (NTNU) in the spring of 2020.

The thesis continues the work from the specialization project which was performed during the autumn of 2019 by the undersigned. Modified versions of chapters 2 and 3 of the project is therefore present in the thesis. Chapter 2 includes a literary study of the most integral datasets and network architectures, as well as previous work within maritime object detection. Chapter 3 contains basic theory regarding deep learning and neural networks in general. It was natural to include these chapters as they are fundamental for both the specialization project and the master thesis.

The dataset used in this thesis was comprised of frames from videos captured with NSMs Seahunter system. The bulk of the images were gathered during a summer internship at NSM, while additional images were added along the course of the specialization project and the subsequent master thesis.

I would like to thank Frank Robin Danielsen from NSM for his help with regards to the capturing of additional videos for increasing the dataset. His support and ideas surrounding the project and thesis has been truly motivational for my work. A massive thanks is also due to my supervisor, Annette Stahl, for her encouragement and guidance. Lastly, I would also like to thank my girlfriend, family and friends for being supportive and understanding during and after the weeks spent drawing ships.

Paul Vik
Trondheim, June 2020

Table of Contents

List of Tables

List of Figures

Abbreviations

ROI	=	Region of Interest
CNN	=	Convolutional Neural Network
ANN	=	Artificial Neural Network
FCNN	=	Fully-Connected Neural Network
AP	=	Average Precision
mAP	=	mean Average Precision
conv layer	=	Convolutional layer
ResNet	=	Residual Neural Network
NSM	=	Norwegian Special Mission
RPN	=	Region Proposal Network
GPU	=	Graphics Processing Unit

Chapter 1

Introduction

1.1 Motivation

Computer Vision has had a massive surge in accuracy and usability in recent years due to the revolutions in deep learning. This rediscovered branch of machine learning has had its renaissance fueled by the increase in processing and computing power. Especially the improvements in graphical processing units (GPUs), and the realization that the functionality of parallel cores were ideal for deep learning, were the key elements of this revolution.

Despite the massive improvements and potential of the technology, deep learning for computer vision still has not been implemented on a large industrial scale. The reason for this is two-fold. First, the state of the art computer vision networks are based on *supervised learning*. This essentially means that the networks learn by being shown multitudes of data such as images, videos or text, and then predict some feature of the data. For this to work, there has to be a *ground truth* or an answer that the network is supposed to predict, such that it may correct itself. Both the collection of the data and classifying it, are the main drawbacks of supervised learning. For it to be effective, it requires massive amounts of data. In addition, the labeling process is also time consuming and monotonous work. Thus, there is a massive incentive to automate this process as much as possible.

Second, the state of the art networks are still fairly unreliable on an industrial scale. While the results are promising, there are still major uncertainties in the application of supervised learning algorithms. The inherent nature of the technology is to find an underlying mathematical model in the data that one can use to predict features in general data. However, unless the original training data contains all possible variations of its data, the final model will be biased towards the training scenarios. The aim of finding an approximately general model with limited training data is an active research field.

Utilizing computer vision for maritime object detection also has several civilian and military usages. For instance, ships are the main artery for trade of physical goods on the

global stage, and a computer vision application could be used for logistic purposes. Other applications is to track the naval traffic in ports and harbors, or for surveillance of naval areas.

This thesis seeks to improve upon the results achieved in the specialization project. The project was initiated by Frank Robin Danielsen at NSM with the aim of researching the applicability of deep learning method to their Seahunter system.

1.2 Challenges

Data Collection

The dataset utilized in this thesis was gathered by capturing video using Norwegian Special Missions (NSM) Seahunter system for the specialization project last year. One of the most obvious methods of enhancing supervised learning performance is to increase the dataset. This is especially true for relatively small datasets such as the this, as additional images will have a larger impact. The process of gathering addtional material was dependant on the operators of the Seahunter system to capture extra footage on their missions. Thankfully, multiple videos were captured during the work on the thesis. However, due to the busy schedule of the operators there was a latency within the data gathering, which ultimately limited the size of the final dataset.

Hard False Positives

A common problem with object detection networks is falsely detecting background objects as the desired classes with high confidence scores. These erroneous detections are known as *Hard False Positives* (HFP). They are hard to filter out, as the model is confident that they are indeed the desired class. The cause of this issue is often that the training data does not include the scenario of the HFPs, and thus the model is uncertain what to predict. The chief issue featured in this thesis is that the models falsely detect small islands as ships. The main dataset included very few images of coastal landscape, as it mainly consisted of ships at sea. These HFPs reduce the overall average precision (AP) score and reduces the robustness of the models significantly.

Mask Annotation

In order to be able to utilize instance segmentation networks such as Mask-RCNN and Cascade-RCNN, one has to have access to a dataset with *mask* annotations. A mask is essentially an outline of the individual objects (instance segmentation), or a group of objects within the same class (semantic segmentation). There exist several publicly available datasets with mask annotations such as COCO and ImageNet. However, one usually gains better performance when the model is trained on samples from the relevant scenarios. Thus, a mask annotated set is required. The process of annotating masks can be performed in multiple ways, depending on what annotation tool one uses. The simplest method is to draw a polygon around the object, by clicking point for point. In any case, this form of

annotation is significantly more time-consuming than bounding box annotation, which is a large incentive to automate this process.

Detection of Small Objects

During the specialization project, it became apparent that the models struggled with detecting smaller vessels. This is an inherent challenge within computer vision, as smaller objects are represented with smaller numbers of pixels, and thus contain less features for the network to learn or detect. In many applications, the detection of smaller objects is not an issue. However, for detection of naval vessels it is crucial, as the camera is mounted on an airplane and the videos is therefore captured a fair distance from the target objects.

Figure 1.1: An example of small objects within the dataset.

Computational Requirements

In order to be able to train instance segmentation networks one requires access to significantly powerful hardware, specifically a powerful GPU with preferably 12 GB of memory. Additionally, it would also be beneficial if the GPU contains so-called *CUDA cores*, which are designed for deep learning applications. Most of the deep learning frameworks such as Tensorflow and Pytorch are specically written with the CUDA library in mind to run efficiently, and having access to a GPU with this functionality will accelerate the training process considerably. The memory requirement is necessary to use images with large resolutions as input. The resolution of the images is important for the detection of smaller objects, as they are represented with a small of number of pixels, and are thus the most vulnerable to resizing of the input images. The latter is necessary if one only has access to a GPU with less memory.

1.3 Aim of the Study

The overarching aim of this thesis is to assess the applicability of deep learning methods to detect naval vessels in a practical environment. A secondary objective is to improve upon the results of the specialization project. In order to ascertain these questions, instance segmentation architectures will be trained and tested on a custom dataset. This category of neural networks was chosen as they are, at the time of writing, state of the art within both object detection and instance segmentation tasks. Instance segmentation networks require mask annotations for the training process. Additionally, the custom dataset will eventually have to be enlarged for the performance to improve over time. Thus, a third objective arise; create a pipeline for dataset creation with efficient mask annotation. Finally, recent studies (He et al. (2018)) indicate that the paradigm of transfer learning within computer vision might be overthrown by training from scratch regimes. In this thesis, both of these training strategies will be utilized to determine which is suitable for smaller datasets.

With these goals in mind, the following milestones were derived:

- Create a pipeline for dataset creation.
- Build upon the existing dataset with additional images and mask annotations.
- Train Mask-RCNN from scratch and with transfer learning.
- Train Cascade-RCNN from scratch and with transfer learning.
- Test the resulting models on a practical test set.

1.4 Contribution of this Thesis

The contribution of this process is a comparison of multiple state of the art convolutional neural networks and comprehensive test results of their performance on a real world dataset in a practical scenario. A mask annotated dataset was generated for this purpose. In addition, a proposed data pipeline for gathering and annotating was made with the aim of simplifying and accelerating further works within this scenario. A slightly modified method for training the networks is proposed as well. The method involves fully training the network with pre-trained initialized weights.

Chapter 2

Background

This section will explain the fundamental work which the thesis builds upon. The layout of the chapter is presented in the following order. First, a short description of each dataset that are either used in this thesis or for pre-training of the backbone networks. Second, a section which explains the architecture of the backbone architectures used in this thesis, and subsequently explanations of the larger instance segmentation and object detection networks. Third, a review of the previous work that is performed in the field. This subsection contains the work done in the specialization project which this thesis builds upon, and other similar work. Lastly, a review of the open-source software which was utilized in this thesis.

2.1 Datasets

There are two key parts in supervised deep learning applications, the network and the dataset. The network performs only as good as the data it is trained on, and thus it is crucial to have large amounts of data available. Thankfully, there are multiple large state of the art datasets that are openly available online.

2.1.1 Publicly Available Data Sets

The following datasets contain millions of images, with large number of classes and subclasses. The common feature between the datasets is that they have been used in competitions, and currently they are used as benchmarks for the deep learning community. Another common usage is that they are used for *pre-training* networks. This means that a network is trained on a subset of these datasets, were the generated model then can be used as a base model which can be fine-tuned by training on a smaller custom dataset. This technique is called *transfer learning* and is a mainstream approach for deep learning applications.

ImageNet

ImageNet (StanfordVisionLab) is one of the largest available datasets. It was introduced in 2009 by Fei-Fei Li and Christiane Fellbaum to the Conference on Computer Vision and Pattern Recognition (CVPR), and from 2010 onwards it has been used annually in computer vision contest called ImageNet Large Scale Visual Recognition Contest (ILSVRC). In total it contains over 14 million annotated images, a million of which are annotated with bounding boxes. ImageNet classifies objects as synonym sets, or ”synsets”, which essentially mean objects which can be described multiple words or phrases. The term synset stems from the predecessor WordNet, which was partly created by Fellbaum. There are over 21,841 synsets in ImageNet.

ImageNet-1k

For most applications and network, 21,841 synsets are far too many to utilize in a practical application. Thus, a standard practice is to use a subset of the total dataset. As the name ImageNet-1k suggests, it contains 1,000 of the highest quality synsets.

COCO

Microsoft Common Objects in Context (COCO) (COCOConsortium) is the primary dataset for instance segmentation, and is also the benchmark test set for state of the art architectures within multiple branches of computer vision. It contains over 330,000 images, over 200,000 of which are labeled. In these images, there are 1,5 million object instances with 80 object categories. COCO contain mask annotations for instance and panoptic segmentation, bounding boxes for classical object detection and key-point annotations for people.

2.1.2 Dataset generated from NSMs Seahunter System

In the specialization project (Vik (2019)) a dataset with bounding boxes was created. The images was gathered by first collecting videos which contained frames including ships, extracting a frame for every third second, and then manually selecting the frames which contained one or more ships. The criteria for selecting a frame was that the objects were clearly visible, i.e. not too small or blurry, or that most of the object was within the frame, i.e. the camera was not too close to the object. After gathering this initial set, the dataset was refined in several rounds by attempting to balance it as much as possible with regards to the size of the objects, the color scheme in the images, the subclass of the object (fishing vessel, container ship, coast guard etc.) and the position of the objects. After multiple rounds of refining, the remaining images were annotated by hand with bounding boxes and classified. Due to time restraints, the naval vessels were only classified as ”ship” or background.

Table 2.1: Overview of the data sets.

Data Set	Number Of Images	Number of Objects
Training Set	$3,703$	$4,272$
Unrefined Test Set	$1,516$	$2,448$
Total Dataset	$5,378$	$6,894$

In table 2.1 there is an overview to the training set, test set and the total dataset. In total there are 5,378 box annotated images from the specialization project, of which 3,703 are dedicated to training, and 1,516 are dedicated to testing. The remaining images were judged to be of too poor quality to be in either set. The training set contains the more refined images of the total dataset, i.e. the images are themselves clearer and the objects are visible in the frame. The Unrefined test set was used as a hard benchmark test set in the specialization project, as it mostly contains blurry or noise images, or the objects themselves are quite small. This irregular split in training and test sets was made in an iterative fashion, where by first splitting the total dataset by the quality of the images, and then training and testing YOLOv3 on the respective sets. It was found that by enlarging the training set with images of less quality from the Unrefined test set the performance decreased, and thus the split remained as specified in table 2.1. The relatively large test set also allows for a more accurate test result. If the performance is good on tough imagery, then it is a better indication of a robust result.

In this thesis, this split between the training and test set will remain the same as the specialization project to be able to compare the results between the respective performances of the two projects.

2.2 Object Detection Architectures

Object detection is a branch of computer vision with the aim of creating a bounding box around the detected objects, and subsequently classifying them. Instance segmentation is a similar branch to object detection, except that in addition they provide a pixel-by-pixel mask around the objects. Due to this overlap, instance segmentation networks works fine for object detection as well. In fact, according to (PapersWithCode) the state of the art network for object detection on the COCO test-dev benchmark, is an instance segmentation network, Cascade-RCNN (at the time of writing). In this section, a number of network architectures and their concepts for object detection and instance segmentation will be explained.

The section begins with the various backbones that are either fundamental or utilized in this thesis. Then it follows up with the instance segmentation architectures as a whole and their important concepts. After that, a review of previous work on maritime object detection, including the most relevant articles and master theses as well as the fundamental work that was performed in the specialization project. Lastly, there are descriptions of the openly available software that is used in this thesis.

2.2.1 Backbones

When building object detection architectures, the computer vision community quickly realized the advantages of building upon the existing object classification architectures. Networks such as VGG-16/19, ResNet-30/50 or the Inception networks were state of the art feature extractors, which with little overhead could be modified to detect objects as well as classifying them.

Today, the common practice is to create modular architectures, with a feature extractor or *backbone*, and the detection part. This way, relatively old architectures such as Mask-RCNN (He et al. (2017)) and even Faster-RCNN (Ren et al. (2016)) can achieve state of the art results by upgrading to a new and better backbone.

A backbone architecture is typically a preexisting network architecture that has been used for object classification. The reason behind this is that this type of architecture has been proven by extensive usage to pick up features in images, and they are relatively easy to build upon. They are most often deep convolutional neural networks, each with their own architectural modifications.

However, the backbone can also be custom made to fit a certain criteria, or match the architecture better. For instance, DarkNet was made to find a better tradeoff between accuracy and speed when used in YOLOv2 (Redmon and Farhadi (2016)), that other preexisting object classification architectures did not provide.

VGG

VGG-16 (Simonyan and Zisserman (2015)) was an improvement on the pioneering convolutional neural network AlexNet from 2012. It continued on the trend of ”the deeper the CNN the better” adding more convolutional (conv) layers, and reducing all the filters to 3x3. The number in the end is simply a notation for how many conv layers there are in the network. VGG-19 added three more layers and performed only slightly better even though requiring more memory. In the competition ILSVRC-14, VGG finished second in classification, but first in localization. The architectures of VGG-16 and VGG-19 are shown in figure (2.1)

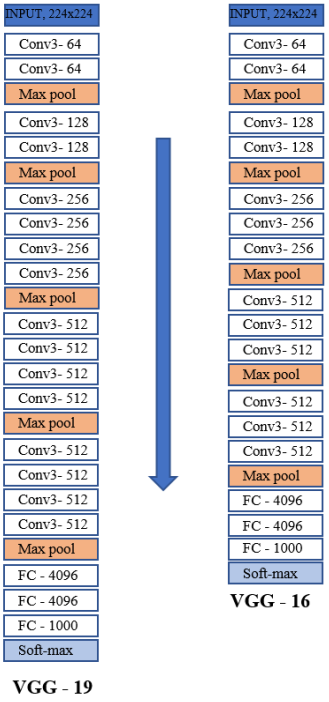


Figure 2.1: The architectures of VGG-19 and VGG-16.

ResNet

The problem with adding more and more layers was that the gradients in the backpropagation algorithm either vanished entirely or exploded in the earlier layers, resulting in that the weights of those layers not changing at all or far too much. This ”Vanishing Gradient” problem prevented adding more layers to gain better performance. The solution was introduced however, in the ILSVCR the year after.

In 2015 there was a ”revolution in depth” of convolutional neural networks when Microsoft won the ILSVRC-15 contest with their Residual Neural Network (ResNet) (He et al. (2015)). The solution to the vanishing gradient problem was introducing ”shortcut” connections. This shortcut connection is also referred to as the Basic Block of the RNN, and consists of shortcut that jumps over two 3x3 conv layers.

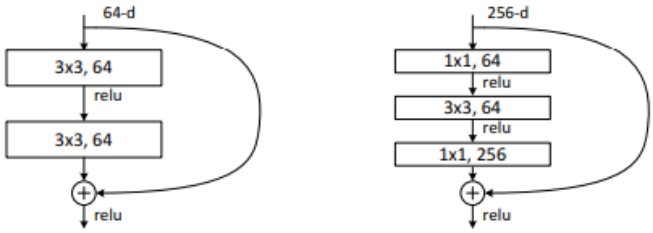


Figure 2.2: The original residual block from ResNet-34 on the left, and the bottleneck residual block from the later ResNet networks on the right (He et al. (2015)).

The architecture of ResNet is split in three parts, where the first and bottom part is the VGG-19, following that is a 34 layer plain network middle part, and finally a 34-layer residual network (ResNet). The naming convention of the ResNets are based on the third part, i.e. ResNet-34 has 34 residual layers, ResNet-50 has 50 residual layers and so on. To reduce the complexity, i.e. reduce the number of parameters, a ”bottleneck” residual block was introduced. This bottleneck block has a shortcut that jumps over three conv layers, where the first and third are 1x1 conv layers and the second is a 3x3 conv layer. The 1x1 conv layer has fewer parameters than the 3x3 conv layer, while it does not degrade the performance too much. With this bottleneck block instead of the basic block, it becomes a 50-layer residual neural network, or ResNet-50. The deeper networks ResNet-101 and ResNet-152 are made by continuing this bottleneck pattern.

The performance of ResNet increases with the number of layers, but obviously so does the computing requirements. According to the paper He et al. (2015), ResNet-50 achieved 20.74 % top-1-error and 5.25 % top-5-error on single model results on the test set of ImageNet from 2015, whereas ResNet-152 achieved 19.38 % top-1-error and 4.49 % top-5-error. Although ResNet-152 is quite a bit larger, it still only improves on ResNet-50 by having approximately 1 % lower error rate.

ResNeXt

In 2016, the term *cardinality* is introduced in Xie et al. (2016), which in essence is an increase in depth of the ordinary residual block in the ResNet architectures. This concept is similar to the idea behind the Inception architectures (Szegedy et al. (2014)), in that they add parallel ”paths” in each block. The number of these paths is what is referred to as cardinality. The difference between ResNext and the Inception networks is that the paths of the latter networks are custom designed for each layer, whereas the paths of the former has the same ”design”, or *topology* as it is referred to in the article.

The concept of cardinality is combined with the traditional idea behind ResNet, which is residual connections and the continued pattern of the residual blocks. The sizes of the ResNeXt networks mimick the sizes of the ResNet architectures (18/34/50/101/152), such that they may be directly compared with eachother. Despite containing more paths in each

block, the number of parameters and FLOPs are comparable. For instance, ResNet50 and ResNeXt50 have $25,5 * 10^6$ and $25,0 * 10^6$ parameters and $4,1 * 10^9$ and $4,2 * 10^9$ FLOPs respectively. Although ResNeXt50 is slightly more computationally expensive, it weighs up for it with better performance than its predecessor.

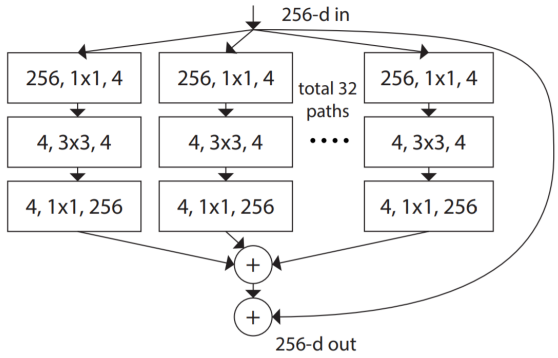


Figure 2.3: An illustration of the ResNeXt block (Xie et al. (2016)).

Feature Pyramid Networks (FPN)

A feature pyramid network is a feature extractor architecture which, as the name suggests, consists of multiple feature pyramids. A feature pyramid refers to feature maps which are extracted from different depths of a convolutional network. Due to the resolution being smaller the deeper its respective layer is, and visa versa larger the closer the layer is to the input, a pyramid-like shape appears. The pyramids are constructed with a bottom-up pathway, which essentially is a standard backbone architecture, such as ResNet, and a top-down pathway. The top-down pathway is generated by first taking the final feature map output from the backbone, and then scaling it up and then combining the result with a feature map from the backbone through a lateral connection. The lateral connections are made between feature maps with the same spatial size. The idea behind this is that the final feature from the backbone are high-level features, which depicts the strongest features in the image. When upscaling this condensed feature map, while combining it with the features from the bottom-up pathway, one achieves feature maps of a higher quality.

It was found that if one uses these feature maps from multiple scales, the convolutional neural networks detect objects of different sizes far better than an architecture which only utilizes one feature map scale. The feature pyramid networks were introduced in (Lin et al. (2016)), and by combining it with Faster-RCNN, Tsung-Yi Li et al. achieved better performance than its regular feature extractor counterparts. FPNs are now standard practice in state of the art detectors, as they provide good results with little to no overhead.

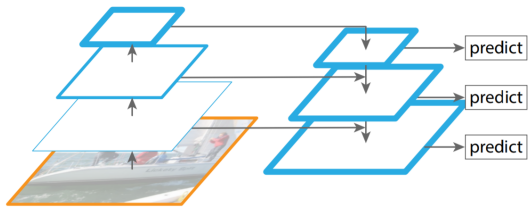


Figure 2.4: An illustration of the Feature Pyramid Network structure (Lin et al. (2016)).

2.2.2 Faster-RCNN

Faster-RCNN (Ren et al. (2016)) was the state-of-the-art object detection architecture in 2015, and was the foundation of both 1st place entries in ILSVRC and COCO competitions the same year. It built upon Fast-RCNN by R. Girshick (Girshick (2015)) by recognizing that using selective search for finding region proposals was the bottleneck for increasing the speed of the network. The solution to this was implementing a shallow convolutional neural network for generating these region proposals, which was aptly named the Region Proposal Network (RPN).

Faster-RCNN can be divided into three parts. The first is the feature extractor generating feature maps, the second is the RPN and the third is a classifier. Shaoqing Ren et al. reuses the concept from Fast-RCNN, where the classifier and the region proposal algorithm share the same feature map from the backbone. This way, one only need to calculate the features once for the entire image rather than for every region proposal. Also, the input to the RPN is far smaller and the features are more condensed, allowing the network to be shallow and lightweight.

The RPN is a fully convolutional neural network, which in this case takes the feature map as input and outputs a set of rectangular region proposal together with an objectness score. The RPN slides a small convolutional neural network over the input, the output of which is then connected to two fully-connected layers. For every sliding step, the RPN predicts up to k different region proposals. k is the number of predetermined reference boxes, called *anchors*. The anchors are centered at the current sliding window position, and consist of a scale and an aspect ratio. The default solution in Ren et al. (2016) is to use three scales and three aspect ratios.

The fully connected layers are for box-regression and box-classification, which outputs bounding box proposals along with their respective classification scores. Non-Max Suppression is then performed to select the best proposals, which is then used in the classifier. The classifier cuts out the regions of interest (ROI) from the original feature map. But as the classifier is a fully connected neural network, it requires equally sized inputs. To assimilate the differently sized ROIs, a technique known as ROIPool is performed. ROIPool essentially divides the ROIs into HxW (e.g. 4x4) grids, and then performs maxpooling for all values inside the grids. The grid itself is normalized according to the procedure explained in Girshick (2015).

When the ROIs are of equal size, they can be put through the classifier, which outputs a predicted box with its confidence score and classification. The confidence score is a measure of how certain the bounding box is to contain an object.

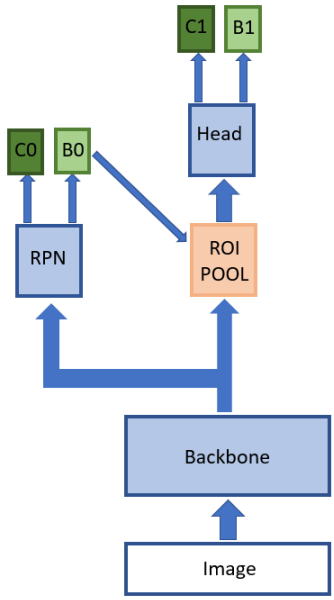


Figure 2.5: An illustration of the Faster-RCNN architecture. B0 are the bounding box proposals generated by the Region Proposal Network, whereas B1 are the predicted bounding boxes. C represents the classifications.

2.2.3 Mask-RCNN

Mask-RCNN (He et al. (2017)) is a deep convolutional neural network, which is based on the architecture of Faster-RCNN. It introduces several changes from the latter network. First of all, it adds a Feature Pyramid Network (FPN) as the feature extractor, but the most important addition is the new branch in the Region Proposal Network for predicting a *pixel-to-pixel* segmentation mask. The new branch in the RPN is a small Fully Convolutional Network (FCN) which is applied to every ROI, and runs in parallel with the box regression and classification.

They also introduce an improvement on ROIPool from Girshick (2015), called ROIAlign. The problem with the former is that it forces the grid to be divided into spatial bins of integer coordinates, that are then quantized by maxpooling. This enforcement of integer coordinates creates misalignment between the features in the feature map and its quantization. This is not especially harmful for classification purposes, but it reduces the quality of the pixel-to-pixel masks drastically.

He et al. (2017) proposes ROIAlign as a solution to this issue. Rather than using integer coordinates on the corners of the grid, it instead calculates the value of each sampling point by performing bilinear interpolation on the surrounding grid points of the feature map. Thus, the corners of the quantization grid does not match the corners of the feature map, but they do match the features they represent. The use of ROIAlign versus ROIPool resulted in a performance gain of 0.011 AP^{bb} when applied to Faster-RCNN.

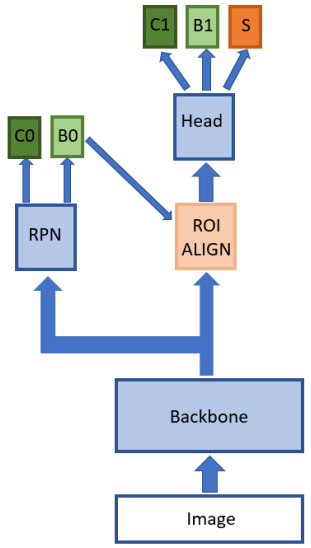


Figure 2.6: An illustration of the Mask-RCNN architecture. B0 are the bounding box proposals generated by the Region Proposal Network, whereas B1 are the predicted bounding boxes. C represents the classifications, and S represents the predicted segmentation masks.

2.2.4 Cascade-RCNN

Cascade-RCNN (Liu et al. (2019)) is a state of the art object detection network. It is built directly upon the architecture of Mask-RCNN, however it introduces multiple improvements on its predecessor. Chiefly, it introduces the *cascade*, which is intended to address the paradox of low quality detections that arise with a low IoU (Intersection over Union) threshold of 0.5, and high quality detections with higher IoU thresholds but degrading detection performance. The concept behind the cascade is to use the output bounding boxes from a Mask-RCNN network, and use those as training input in a second classifier head. As seen in figure 2.7, it continues on with this pattern with multiple classifiers and RPN. In other words, it keeps the first stage from Mask-RCNN intact, but it adds multiple ”second” stages. For every new stage, the IoU threshold is increased, and thus one is able to make high quality predictions with high accuracy.

Cascade-RCNN does not use hard-negative mining. Hard negative mining is a technique to suppress the number of hard false positives, by using false positives from a previous training iteration in the next iteration. The inherent nature of the cascade is similar, but

does not use this technique explicitly. As the stages in the detector stage use the bounding boxes from the previous stage $n-1$, the input to stage n contains *close false positives* which are essentially false positives with slightly worse IoU score than the threshold. This results in gradual improvement in detection quality from stage to stage.

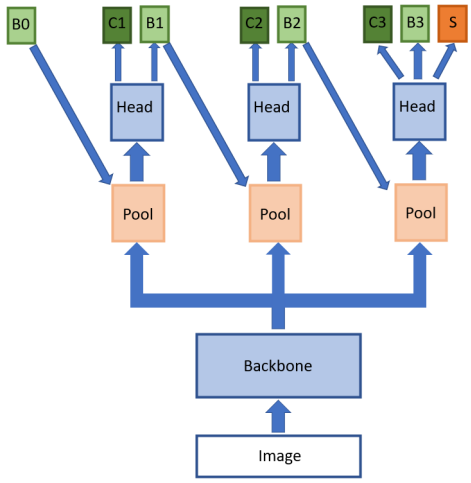


Figure 2.7: An illustration of the Cascade-RCNN architecture. B0 are the initial bounding box proposals, whereas B1 are the predicted bounding boxes. C represents the classifications, and S represents the predicted segmentation masks.

2.2.5 Training Strategies

Transfer Learning

The solution to the dependency on large data sets is often first to train the network on a large publicly available data set of similar objects, and then *fine-tune* the network on the smaller and more specific data set. This process is known as *transfer learning*, and is a common approach for training on smaller data sets. The benefits of transfer learning was partly introduced with the Region-CNN architecture (Girshick et al. (2013)) and in Donahue et al. (2013). The fine-tuning involves freezing the majority of the layers, and only train the last layers of the network. Thus, training the network takes a lot less time as only a small part of the network is altered. In addition to this, one reduces the chance of *overfitting*. A frequently used approach has been to train the backbones on the large classification set ImageNet, and subsequently freezing these layers and fine-tune the remaining layers on a smaller custom dataset.

There is a prerequisite that has to be fulfilled for one to be able to use transfer learning. The large data set that the network is originally trained on, has to contain somewhat similar objects or features to the objective in the smaller data set.

Training From Scratch

In the last years, transfer learning has been a paradigm within the computer vision community. It and was long the default method of training state of the art, two stage architectures, as it gained good accuracy and quicker convergence. However, in late 2018 this approach was challenged by a paper by the computer vision pioneers Kaiming He, Ross Girshick and Piotr Dollár from Facebook Articial Intelligence Research (FAIR) (He et al. (2018)).

The authors suggest and show that training from scratch is a potent opponent to the transfer learning approach. The paper provides the following three observations. First, that utilizing pre-trained weights and subsequently fine-tuning the network speeds up the convergence of the models. Second, this does not necessarily provide better regularization than training from scratch. Third, they observed that training from scratch noticeably increased the AP score for high box thresholds.

In this thesis, both of these training strategies will be applied to determine which is the most suited to smaller datasets that contain under 10,000 images. The expressions ”trained from scratch” an ”fully trained” will be used interchangeably.

2.3 Previous Work

There have been multiple studies regarding the application of computer vision and deep learning to detecting naval vessels. The variation in the field often stems from a difference in the imagery where the object detection is applied. For instance, satellite imagery have been used to track international trading vessels, and in other cases imagery from harbor level have been used to detect vessels coming in and out of their respective areas. Another separator is the actual use-case of the model. Some fields have a strict real-time requirement, which limits the use of state of the art architectures as they are far too computationally expensive, whereas other fields have no such limitations.
In this section, a few relevant key studies will be reviewed, as well as a description of the fundamental work performed in the specialization project.

The Specialization Project

The specialization project was performed in the autumn of 2019, with the aim of applying object detection networks to detect naval vessels. With this in mind, a dataset using videos from NSMs Seahunter system was created, and two architectures, YOLOv3 and Mask-RCNN, were tested on it. The dataset was created by manually choosing appropriate frames from the video footage, and subsequently annotating them with bounding boxes and a class, ”ship”. Due to constraints in computational power, only a smaller variant of YOLOv3, YOLOv3-tiny was trained fully on the training set. However, several variants of YOLOv3 were trained using transfer learning, by utilizing weights pretrained on COCO. Mask-RCNN was not trained due to both the lack in available hardware, but also the lack of mask annotations in the data set. The performance of Mask-RCNN was judged by using inference with COCO pretrained weights on two custom test sets. The first was a simple

test set of 365 images, whereas the second was a harder test set, consisting of 1,516 images containing small or blurry objects. Mask-RCNN achieved a score of 0.73 AP_{50} on the smaller test set, and an 0.384 AP_{50} on the larger test set. YOLOv3-tiny surprisingly achieved the best results, gaining 0.971 AP_{50} on the smaller test set, and 0.385 AP_{50} on the larger test set. The quality of bounding boxes of Mask-RCNN were better however, and keeping in mind that it was not trained on the dataset, this result is not that remarkable. However, it is still interesting that the smaller network YOLOv3-tiny, performed better than its larger variations, YOLOv3 and YOLOv3-SPP, when trained fully rather than fine-tuned.

The specialization project leaves a couple of questions to be answered. For instance, how much would Mask-RCNN improve when trained on the training set? How does these methods compare to the actual state of the art methods at the time of writing? Is it possible to create an efficient dataset creation pipeline to reduce the human overhead?

2.3.1 Other Work Within Maritime Object Detection

Automatically detecting maritime objects has several usages for both military and civilian purposes. Ships being the main transportation for goods in worldwide trading, there is a large incentive for detecting ships for logistic purposes. Thus, with the explosive growth of object detection using deep learning, applying it to naval vessels is a natural step forwards. In this task, only previous deep learning methods in this field will be looked at, as the motivation for this project is determine the applicability of deep learning methods for ship detection.

In the master thesis Grini (2019) two object detection methods, YOLOv3 and Single Shot Detection (SSD), are proposed for detecting maritime objects in Trondheimsfjorden. The data set was generated by taking photos of moving boats in the fjord as well as moored ones. The angle of the photos are therefore from a boats viewpoint, i.e. from ground level. The purpose of this thesis is to verify whether the object detection is reliable enough to be applied on an autonomous vessel. To be able to use an object detector in a control system one has to fulfil strict requirements, including real time performance and high confidence in its output. This rules out the heavier, state of the art object detection architectures like Mask-RCNN. With YOLOv3, Grini achieved a high score of 90.8 % AP on a test set of boats in the fjord, but it was reduced to a score of 70.7 % AP when the test set only contained moored boats. The performance of SSD was only slightly worse on boats in the fjord (87.6 % AP), but performed severely worse on the moored boats (58.6 % AP).

In Shaodan et al. (2019) an improvement on Mask-RCNN for better detection of offshore small ships is proposed. Their main contribution is optimizing the RPN loss function as well as the mask generation algorithm, gaining a small increase in Average Precision over the default Mask-RCNN.

The paper Nie et al. (2018) applies Mask-RCNN for the inshore ships, and proposes *Soft-Non-Maximum Suppression* (Soft-NMS) for detecting objects that are in close vicinity of each other. Their data set is generated from satellite imagery of crowded ports.

2.4 Open-Source Software

The deep learning community has so far been exemplary in their scholarly approach of sharing results and information. State of the art networks are made publicly available online, such that the technology may grow and evolve in an international effort. Thankfully this includes massive technology corporations such as Facebook and Google, which have made much of their research available. In addition to the research papers, the community also provide excellent deep learning software. It is common practice to attach a Github repository for their code for reproducing results, or building upon it. This section includes some of the open source software that is utilized in this thesis.

2.4.1 Facebooks Detectron2

The Detectron2 framework (FAIR) contains implementations of multiple state of the art networks such as Mask-RNN and Cascade-RCNN. It was created by Facebook AI Research (FAIR) group. The framework contains an extensive ”model zoo”, which essentially are the backbone weights of ImageNet and COCO pretrained models, enabling transfer learning to a large extent. As it is a popular framework among the computer vision community, there is a lot of support around the tools. There are also a simple tutorial notebook for getting started with the framework, which reduces the barrier of entry significantly. The framework itself is based on Pytorch, and the package requirements are relatively few compared to other computer vision frameworks. All of this makes Detectron2 very user-friendly and easy to build upon.

2.4.2 Annotation Tools

There are multiple tools available for the annotation process. Some include more functionality, whereas others are easier to use. The main annotation tool utilized in this thesis was VGG Image Annotator (VIA) (VGG). This is popular and simple annotation tool, which was developed by the Visual Geometry Group (VGG) from the University of Oxford. It supports multiple types of annotation, including bounding boxes and polygons, and it can store the annotations in several of the most common formats, including csv, json and the COCO format. In the latest via-2.0.9 version however, there is a bug when saving the project in the COCO format, where the annotations does not include their respective ”category id”, which is the class of the object. At the current state this requires a custom script to convert their annotation to COCO format, making it harder to use for some applications which require this format.

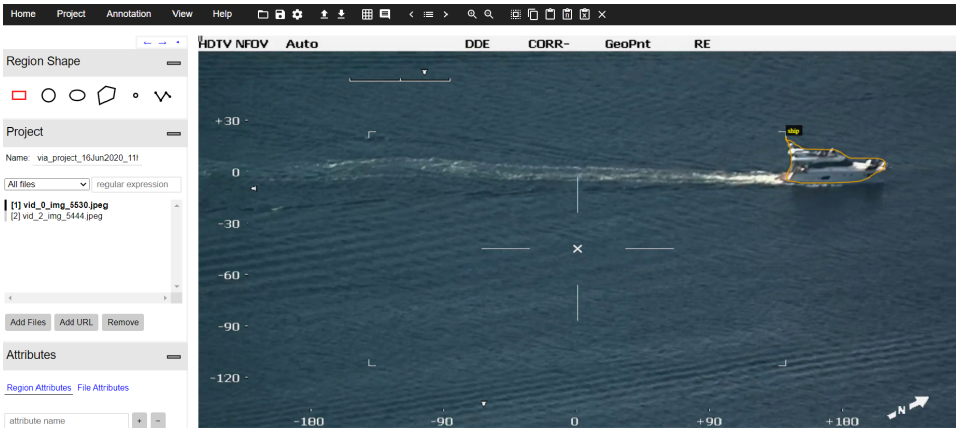


Figure 2.8: The VGG Image Annotation tool.

Computer Vision Annotation Tool (cvat) (Intel) is an annotation tool which is specifically designed for computer vision tasks, and is particularly suited to annotate videos. It is made and supported by Intel. Cvat includes the functionalities of interpolation of bounding boxes between frames, and semi-automatic annotation by using user-specified pretrained deep learning models. This accelerated the dataset creation process significantly. Cvat also has the possibility of dividing the annotation project into tasks, which can be assigned to separate users. This is very helpful for larger computer vision projects where the annotation process is performed in teams. The downside to using cvat is that the barrier of entry is slightly larger than VIA, as it requires more setup. However, the added functionality likely makes up for the slower start.

Chapter 3

Deep Learning

The concept behind deep learning, neural networks and perceptrons, was first introduced in the late 1950s by Frank Rosenblatt (Rosenblatt (1958)). The perceptron was a node containing numerical values. A neural network was formed by connecting multiple perceptrons together. Rosenblatt introduced both single and multilayer networks. The purpose of the networks was to create a linear classifier. Despite its old origins, it was not able to reach its full potential until recent improvements in hardware and software.

Today, deep learning has a wide variety of applications. From its infancy in binary classification, it has now been used in fields such as speech recognition, computer vision, natural language processing and autonomous vehicles. It has achieved superhuman performance in areas which were thought to be singular human enterprises. In 2017, AlphaGo, a deep learning algorithm developed by DeepMind, beat the current world champion in Go, Ke Jie. DeepMind has also had success with AlphaZero, its chess equivalent, which is most likely the best chess algorithm worldwide.

The purpose of this chapter is to establish and explain certain basic principles in deep learning that will be used later in the report. It will not go in depth about the algorithms that are involved in the networks, but will rather give an overview and introduction to the terms and expressions that are frequently used in deep learning literature.

3.1 Supervised Learning

Machine Learning is split into three main branches, supervised, unsupervised and reinforcement learning. The key difference between the three is simply that the first requires data to train, the second detects a model within the data independently, whereas the latter trains by exploring an environment to find the optimal path. The three strategies are inherently different, however deep learning has been applied to all branches, and there are pros and cons to all disciplines.

A typical appliance of supervised learning is in computer vision, where one normally utilizes a dataset of images. The aim of computer vision is to detect features within imagery, whether it is classification or detection of objects. The concept of supervised learning is easy to grasp, as it is very human in its approach. A supervised deep learning network learns by being shown entries of the dataset, predicting the outcome, comparing the prediction with a *ground truth* and subsequently correct itself by a process known as *backpropagation*. In an object classification application, each image would have to be *annotated* with its class, whereas in an object detection application each object would have to be annotated with both its class and a bounding box to indicate its position. The process of gathering images and annotating is costly, and is thus the greatest weakness of this branch. However, for computer vision tasks, supervised learning is the basis for current state of the art convolutional neural networks.

3.2 Computer Vision Tasks

First of all, the primary modes of object detection has to be defined. The first of which is *object classification*, where the task is to classify an object that is in the image. The most common output of an object classification system is a vector of class probabilities, i.e. each element signifies the probability that the image contains a certain class. Thus, the element-wise sum of the vector should be normalized. The highest probability represents the classification of the network. The constraint of this is that it can only classify one object in an image.

The natural development is to locate objects in the image as well as classifying them. This objective is coined *object detection*. The output of the detection itself is most often a *bounding box* with a corresponding *confidence score*. The bounding box is a rectangular box which encases the object.

For several purposes, it is beneficiary to have a continuous outline of areas on the image. *Semantic segmentation* does exactly this. It classifies areas of the image, and draws a *mask* around continuous regions with the same class. Examples of this would be if one wanted to detect oil spills in the ocean, or cracks in a wall. However, the masks does not distinguish between different instances of a class. A group of people would be portrayed as a large blob.

The next iteration is to generate masks around each individual instance. A continuous mask around objects are obviously a more accurate representation of an object than a simple rectangular box. The objective of locating, classifying and drawing masks around individual objects is called *instance segmentation*.

3.3 Traditional Methods

Preceding Artificial Neural Networks, object detection tasks relied a lot on the intuition of the engineers. There has been developed several methods for finding objects in images, however most are highly specific. Examples of earlier approaches have relied on finding key points in the image, such as corners, edges and flats of objects, and matching them with a template of the object one wishes to find. This is known as object recognition rather than object detection, but it has several similarities to one-class object detection. These techniques have several flaws. For instance, they rely too heavily on the template having the same characteristics as the objects in the images. Basically, it can't be used for detecting classes, merely specific objects within that class.

Other approaches includes using a sliding window with different scales for localization and a combination of Histogram of Oriented Gradients (HOG) and Support Vector Machine (SVM) for classification. HOG divides a region (the sliding window region) into a $S * S$ grid, and calculates the dominant gradient in each grid. The collection of gradients can be used as a signature for a given shape. This signature is then used by the SVM to classify it. The problem with sliding window is that it is very computationally demanding, as it has to compute all the scales over the entire image. Also, as the scales are designed to match a specific class (for instance a pedestrian would have a vertically elongated rectangle and a car would have a horizontal rectangle), the processing time would increase with every class. This localization technique is also utilized in combination with deep learning in the architectures Region-Convolutional Neural Networks (RCNN) and Fast-RCNN.

A simple segmentation method that could also be applicable to this problem is thresholding. This technique is based around objects having different lighting or color than the background. First, change the image format into greyscale, and then simply color every pixel below or over a certain greyscale threshold black. In the instance of ship detection, the vessel will often have a darker or lighter color then the ocean, and then could easily be located within the image. However, this is a trivial and highly error prone technique, as ships could obviously have different lighting depending on the time of day, or be in the vicinity of other dark objects.

3.4 Deep Learning Methods

Deep learning refers to the usage of Artificial Neural Networks (ANN). While neural networks has existed since the 1960's, it was not until the last decade with the advances of Graphics Processing Units (GPUs), that it became prevalent in object detection tasks. This was due to the immense computing requirements that training such a network has, which was not possible to meet until recently.

The full impact deep learning will have on society is yet to be determined. However, there are several interesting areas where deep learning has been applied. It's strength is largely drawn from finding complex models that humans can not find. Where traditional methods are dependent on human intuition and modelling, deep learning has the capability to find very complex structures in data without those constraints.

At the time of writing, the technology is at the edge of human capabilities, and is on it's way to surpass our capabilities. It can be applied to detecting objects, generating images and finding mathematical models among other things. There exists more traditional solutions for all of these areas, but deep learning has shown great promise and in many cases already surpassed the previous technologies.
The basic principles of deep learning are explained in the following chapters.

3.4.1 Artificial Neural Networks

The term "Artificial Neural Network" refers to the similarity it has with biological neurons in the brain. An artificial neural network is composed of layers of nodes containing a value. A neural network where each node in one layer is connected to every node in the next layer, is known as a "Fully Connected Neural Network", and is the simplest form of neural networks. The term "deep learning" stems from the number of layers in the network. The more layers it has, the deeper it is. Each network consists of an input layer, where each node represents a parameter value, for example a pixel value in an image, connected to a number of "hidden" layers. The last of which in turn is connected to an output layer where each node represents the predictions of the network. The number of hidden layers is a parameter to be tuned for optimal results. Each connection between nodes is weighted by a value, simply referred to as "weights". The layout of the network, i.e. the number of hidden layers, shapes of the layers etc., is referred to as the architecture of the network. Each architecture therefore has a specific number of weights.

The value in the next node is determined by a sum of the values from its connected nodes in the previous layer multiplied with their respective weights. Each node in the hidden layers has an *activation function* that essentially decides whether the node should "activate" or not. The input of the activation function is the sum of the values in the previous layer, and the output is usually a normalized value.
These calculations are done for every node in the network, until a prediction is made on the values in the output layer.

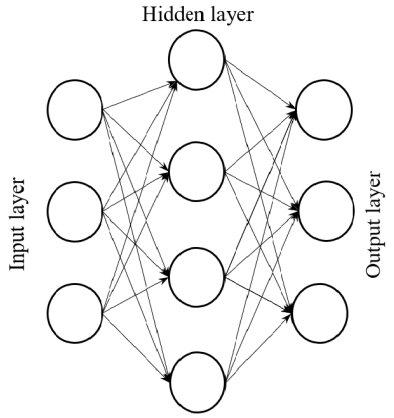


Figure 3.1: A simple fully connected neural network.

Training a Neural Network

When *training* a neural network, one refers to the process of adjusting the weights iteratively until the network hopefully produces credible results. In the initiation of training, the weights are set randomly. Training of the weights is then done by first performing a *forward pass* on the network, which essentially means to calculate the output of the current network, and then calculating the *loss* by passing the output and the target value into a *loss function*. The loss function returns a value that essentially represents the difference between the target and the prediction of the network, also called the *loss* of the network. The loss function is decided according to the task of the network. For instance, a common loss function is mean-squared-error, which works well if the task is to predict a value. Another is Binary Cross-Entropy for binary classification tasks, or Cross-Entropy for classification tasks with more than two classes. More complex tasks such as object detection requires more advanced loss functions to represent deviancy in position as well as class.

Subsequently, the *backpropagation* algorithm is performed to alter the weights of the network. Basically, the algorithm optimizes the weights on a layer-by-layer basis (from last to first) based on the output of the loss function. Doing this for every sample in the training set is called an *epoch*. The number of epochs is another hyperparameter to be tuned.

Overfitting and Underfitting

Overfitting is a common problem when utilizing supervised machine learning. The problem itself arises when the generated model is overly trained on too few training samples, such that the predictions fit too well on the training set. Essentially, the model predicts the ”noise” in the training set as ground truths, and thereby the generated model performs worse on images it has not seen. In machine learning in particular, overfitting is sometimes also referred to as ”overtraining”. Underfitting, or ”undertraining” is the exactly opposite. The model is not trained for long enough, and therefore it does not manage to find the

underlying pattern of in the data set.

Overfitting is the more common of the two, as underfitting is more easily remedied by simply training the model more. There are several strategies to counteract overfitting as well. The data set is most often divided into three parts, the training, validation and test set. The training set is obviously for training the model, and the test set to evaluate its performance. The validation set however, is used to measure the performance of the model while training, and can therefore be used to detect overfitting. If the loss on the validation set starts to rise then it is most often an indication of overfitting, and one can stop the training process. This is called *early stopping*.
Overfitting can also be avoided by simply adding more data entries, or more sophisticated cross validation techniques.

3.4.2 Convolutional Neural Networks (CNN)

A convolutional neural network is fundamentally different to a fully connected neural network (FCNN). The main weakness with FCNNs is that the spatial features in an image are lost in the network, as no node has any information what values other surrounding nodes has. This has severe consequences in classification and detection tasks where the relative positions of features are vital. The introduction of CNNs proposed a solution to this. Rather than slicing up the image into a long list, the CNN keeps the shape of the image, and slides a filter over the image. The filter is a matrix of weights where each element is multiplied with a Blue-Green-Red (BGR) value in a corresponding grid, and then the sum of these multiplications is stored in a new matrix called a *feature map*. This filter slides from left to right over a set stride, and then continues from top to bottom. Each hidden layer is then replaced from lists of nodes to such feature maps, which subsequently are subjected to further filtering in the next layers. After the network has been trained, one can tell by visualizing the different feature maps that they have picked up different features of the image. The final output is a condensed feature map of the most apparent features in the image, which then can be connected to a FCNN for detection and classification.

CNNs have been proven to be far more efficient and accurate than FCNNs. A typical filter is a matrix of 3x3 which has 9 weights, whereas a FCNN with a BGR image of size 1280x720 would have $1280 * 720 * 3 * N_{nodes}$ weights only in the first layer. The main takeaway is that the size of a CNN network is independent of input size, whereas the size of a FCNN would grow immensely with a larger input. Although it is normal to have several filters per layer, it is still far more efficient to use a CNN for non-trivially sized images, as the number of weights affects how long the training and inference takes. Convolutional layers are currently the dominant building block in state-of-the-art architectures for image classification and object detection.

The pooling layers work in a similar fashion to the convolutional layers. They consist of extracting regions from the image (for instance a 2x2 matrix with stride 2x2) in a sliding mode pattern, and then performing an operation on them. The operation is most often returning the highest value in matrix, and this layer is therefore known as a *max pooling* layer.

Effectively, introducing pooling layers reduces the number of parameters in the architecture, and thus cutting the computational cost.

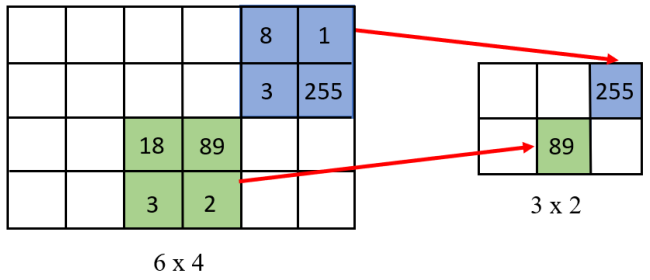


Figure 3.2: Max pool layer: A simple illustration of how a max pool layer with a 2x2 kernel with 2x2 stride works.

Batch Normalization

Batch normalization was first introduced in (Ioffe and Szegedy (2015)) as method of accelerating the training process of deeper networks. The idea was to apply the same form of normalization as is performed during the pre-processing stage at the input layer to all the other layers within the network. The normalization itself is performed by aggregating the values in each layer of all the entries within each batch, and storing the mean and standard deviation of each layer in separate neurons and appending them to their respective layer. Thus the actual values stored within each neuron is substantially reduced, which in turn reduces the amount the neuron values are shifted. Historically, this has enabled quicker training of larger networks, and even achieving better performance.

Batch normalization introduced a problem when utilized in computer vision applications. To use batch normalization effectively, one is required fulfill certain assumptions, the main of which is a sufficiently large *batch size*. In computer vision, one often has a large input tensor, as the data entries are often high resolution images, and due to memory constraints one cannot afford larger batch sizes. (Wu and He (2018)) suggests *group normalization* as a solution to this issue. Rather than computing the mean and standard deviation of one channel in all the of the same layer within a batch, they compute them in *groups*, which are sets of channels within one layer in one data entry. Thus, the group normalization is independent of batch size, and allows for deep computer vision networks being trained from scratch.

3.5 Data Augmentation

Another option if one has a small data set is to increase it by creating synthetic data. This can be achieved in a myriad of different ways. One simple, yet effective strategy is *flipping*. Simply flip a percentage of the images in the training set, and voilá, one has a lot more images. Caution is advised when utilizing this strategy. If the images that are flipped

are of lower quality or contains noise of some kind, one risks ending up with worse results than before. Other trivial augmentation methods include rotation, translation and changing the coloring strategy.

A more modern and deep learning inspired augmentation approach is synthetically generating more images by using the data set in another neural network. This solution is far more complex than flipping, but it might not be more beneficial. If the original data set already contains noise, or is of lower quality, it is likely that the generated images will contain similar disadvantages. Also, this strategy would require far more expertise than the other ones.

3.6 Performance Metrics

Measuring the performance of Artificial Neural Networks can be quite tricky, and each metric needs to taken with a grain of salt. For classification tasks, it is a simple matter of the percentage of correct predictions, whereas for object detection tasks such as the one at hand, one has to take a metric of localization into account. Specifically, how well does the trained model both find and classify objects in an image.

A few definitions are necessary in order to express the following metrics. The Intersection-over-Union (IoU) function is used to determine whether a predicted box actually matches an actual box containing an object (ground truth box). It is defined as:

$$IoU = \frac{Intersection}{Union} \tag{3.1}$$

Where Intersection is the shared area of the two boxes, and Union is the total area of the two subtracted by the Intersection. The IoU function returns a value between 0 and 1, where 1 indicates a perfect overlap.

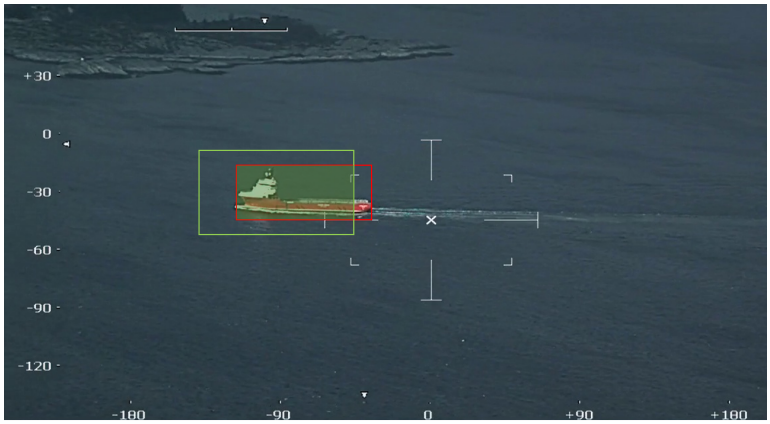


Figure 3.3: A depiction of how the Intersection-over-Union function works. The red box is the ground truth box, and the green is the predicted box. The green area is the intersection of the two.

True Positives (TP) are predictions where the predicted bounding box has an Intersection-over-Union (IoU) value over a certain upper threshold with a ground truth box and the correct classification, i.e. correct predictions. False Positives (FP) are predictions which have an IoU value below a lower threshold with all the ground truths or the wrong classification, i.e. wrong predictions. A False Negative (FN) is when a ground truth box has no predictions, i.e. the model has missed an object.
The Precision of a model is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

The Recall of a model is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

3.6.1 Average Precision (AP)

Average Precision has become the default metric which state-of-the-art models in academia compare each other with. There are several different methods to calculate the AP, as it has been continually built upon to achieve an as accurate metric of performance as possible. A common definition is the 11-point interpolation. Which is defined as:

$$AP = \frac{1}{11} \sum_{r \in 0.0,\ldots,1.0} AP_r \tag{3.4}$$

$$AP = \frac{1}{11} \sum_{r \in 0.0,\ldots,1.0} p_{interp}(r) \tag{3.5}$$

where

$$p_{interp}(r) = \underset{\tilde{r} \geq r}{max}\ p(\tilde{r}) \tag{3.6}$$

The reason it's called the 11-point interpolation is that one divides the recall interval into eleven parts $(0.0, 0.1, ..., 1.0)$. It's also possible to split the interval into smaller parts for further precision in the metric. With model trained for detection of several classes, one calculates the AP for every class and take the average of them. This is called the *mean Average Precision (mAP)*.

To be able to compare performance between different networks, they have to be tested on the same data for the Average Precision to have any comparable meaning. The standard test set has varied over the years. For regular object detection, ImageNet and Pascals VOC2007 and VOC2012 have been used to determine the strength of models. However, the current state-of-the-art instance segmentation models has since 2015 been tested on Microsofts COCO set. The introduction on the latter set has brought with new iterations of the AP metric. For instance, the practice is to vary the IoU threshold for True Positives. The primary variant is to take the average of ten AP values with IoU values from 0.5 to 0.95 with steps of 0.05. The strict variant is to only have an IoU threshold of 0.75, notated as AP_{75}. The Pascal VOC metric (which is also used when determining performance on COCO) has an IoU threshold of 0.5, notated as AP_{50}.

It's worth noting that there is a newer form of Average Precision for masks. Masks are outlines of each detected object, and are the result of Instance Segmentation networks. The mask AP replaces the IoU function for calculating the overlap with a more sophisticated area function. Bounding box AP is still a significant performance metric within both instance segmentation and pure object detection applications, as it allows for comparative results despite the different architectures.

Only bounding box AP will be taken into account in this task, as the goal is to compare an architecture that produces just bounding boxes with an architecture which produces both bounding boxes and masks.

Chapter 4

Dataset Creation

Supervised deep learning networks are state of the art methods in computer vision at the time of writing. The main crux of supervised learning is the requirement of large amounts of annotated data. While there are large datasets publicly available for numerous practical purposes, there is most often a need to fine tune the model on data specific to each area of use. To be able to do this, data must be collected and annotated for such scenarios. Thus, there is an incentive for efficient data gathering and labeling and developing a strategy for performing both of these tasks. It should also be easy to expand the dataset when new data is available.

The purpose of this section is to determine which available methods are the most beneficial for streamlining and accelerating the dataset creation and supplementing processes. Primarily, a pipeline using the VIA tool is proposed, whereas a secondary pipeline using cvat is looked at.

4.1 Choice of Annotation Tool

There are multiple excellent annotation tools that are openly available. The annotation tool VIA was chosen as the standard tool in this thesis as it was utilized to create the original dataset in the specialization project. At the time, this tool was recommended by multiple sources. The chief of which was Matterports implementation of Mask-RCNN (Matterport), which highlighted this tool as containing most of the crucial functionality, as well as being simple to use. The json output format of VIA also matched the dataloader of Matterports implementation. As the existing dataset from the specialization project was written in the VIA format, and the fundamental code for augmenting the dataset was written with this format in mind, it was natural to keep developing the pipeline using this annotation tool.

4.2 Dataset Pipeline Using VGG Image Annotation Tool

The following sections will describe the dataset creation process performed in this thesis. Most of the functionality surrounding this process was built around the VIA tool. The tool specialized functionality include semi-automatic mask and bounding box annotation, dataset augmentation scripts and extraction scripts for creating subsets of the data set. The code uses the specific dataset format of the VIA tool as a basis, which makes it incompatible to other annotation tools without a transform. The VIA format is also simple to understand and to build upon.

4.2.1 Gathering Data

The process of gathering data for this thesis was primarily dependent on an operator capturing video footage using the Seahunter system, of which images could be extracted. During the course of the thesis, no schedule for capturing such footage was established, and the process relied on the operator having both sufficient storage space and time to collect it. This introduced an inherent latency in the data gathering phase of this process. Thankfully, as the dataset from the specialization project was well established, the additional footage was not critical initially for training the networks, however it was crucial for gaining better performance.

Over the course of the thesis, a total of 14 suitable videos were gathered to supplement the dataset. Each of the videos contained multiple naval vessels in different environments. Some videos included footage from a scenario which was underrepresented in the previous dataset, namely video of objects close to the shoreline or islands. The previous dataset contained mostly images of objects at sea or far away from land. The new images posed a challenge as the models trained on the old dataset often would predict the islands to be ships. These false positives are known as hard false positives, as the false predictions have high confidence scores. A naive, but effective approach to solve this problem, is to simply add more images of this scenario to the training set. However, this solution does not scale very well to other scenarios, as it requires imagery from all possible environments to detect objects in them.

There are clear benefits to establishing a proper data gathering strategy, and in future such routines should be made to improve performance over time.

Image Selection Process

The videos were split into one frame for every second. The generated images were subsequently subjected to multiple rounds of refining. The refining process simply involved picking out the images of higher quality which contained either islands, close shoreline. The requirements for determining the whether to include an image or not was the following;

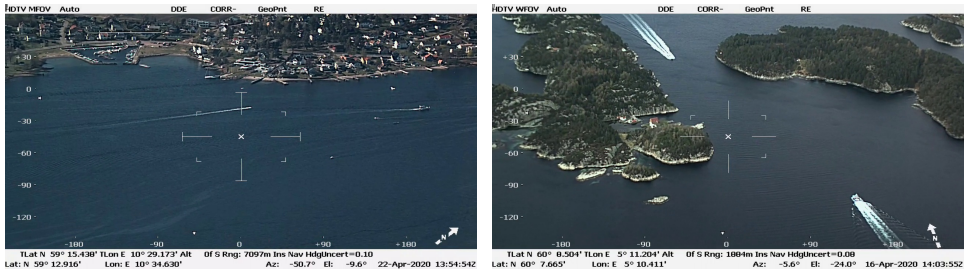


Figure 4.1: A selection of shoreline images from the dataset.

- The most of the features of at least one ship is visible within the image.
- The image is close to the shoreline.
- The image includes one or several small isles.
- The image is not blurry nor too dark.

Determining the most suitable images for expanding the dataset is a difficult task, as it is uncertain what criteria results in the most effective learning. The aforementioned requirements served as a guideline to select each image. If an image satisfied either of the first three requirements, it was deemed to be sufficient for the dataset. This process is hard to automate, as it is difficult to determine automatically the quality of the image and the objects within the image.

4.2.2 Annotating the Data

The topic of automatic annotation arose when faced with the labor intensive process of mask labeling. The previous dataset was annotated using bounding boxes, and to be able to use instance segmentation networks one had to relabel all of the objects in the existing dataset. Drawing masks using the ”polygon” tool in VIA involves drawing an outline by manually clicking each vertex around each object. Drawing bounding boxes on the other hand, only requires clicking on the top left and bottom right corner of the box around each object, making the former process several times more time consuming than the latter. Thus, there is an even larger incentive to automate the mask annotation process than the bounding box annotation.

Annotation Format

The format which is most often compatible with computer vision libraries such as Detectron2 is the COCO format. It is a standardized format of storing the annotations for multiple computer vision tasks such as object detection, instance segmentation, key-point detection, image captioning and panoptic segmentation. The structure for object detection and instance segmentation is virtually the same, and is shown in figure 4.2.

```
{
   "info": info,
   "images": [image],
   "annotations": [annotation],
   "licenses": [license],
}

info{
   "year": int,
   "version": str,
   "description": str,
   "contributor": str,
   "url": str,
   "date_created": datetime,
}

image{
   "id": int,
   "width": int,
   "height": int,
   "file_name": str,
   "license": int,
   "flickr_url": str,
   "coco_url": str,
   "date_captured": datetime,
}

annotation{
   "id": int,
   "image_id": int,
   "category_id": int,
   "segmentation": RLE or [polygon],
   "area": float,
   "bbox": [x,y,width,height],
   "iscrowd": 0 or 1,
}

categories[{
   "id": int,
   "name": str,
   "supercategory": str,
}]

license{
   "id": int,
   "name": str,
   "url": str,
}
```

Figure 4.2: The structure of object detection/instance segmentation in the COCO data format.

filename	file_size	file_attributes	region_count	region_id	region_shape_attributes	region_attributes
str	int	dict	int	int	polygon/ box	dict

Figure 4.3: The structure of object detection/instance segmentation VIA csv format.

Unfortunately, the functionality for exporting the annotations in the COCO format within the VIA tool did work optimally at the time of writing. Instead, the annotations were stored in a csv file in the format in figure 4.3. This was the primary format that was used while developing the dataset annotations, as it required far less storage space than the COCO format.

Initial Automatic Annotation

To achieve semi-automatic mask annotation of the ships in the dataset, one has to utilize a previously trained instance segmentation model to predict the outline of objects in the images, which hopefully are good enough to be used as labels in the new dataset. The choice of instance segmentation network fell upon the Matterport implementation of Mask-RCNN which was utilized in the specialization project. It achieved decent results in the project, and would serve as a decent starting point for the automatic annotation process. The specific model that was chosen in this implementation for automatic annotation was Mask-RCNN with ResNet101 as backbone pretrained on COCO 2017.

After writing a script which would run inference on a set of images, the resulting mask predictions from the network were extracted and transformed to a polygon format, and written to the VIA csv format. The idea behind this was to be able to easily get an overview and augment the automatic annotations within the VIA tool, as the annotations would show up as if manually annotated. Within the tool itself, it was easy to change or delete the annotations if they were inaccurate or distorted.

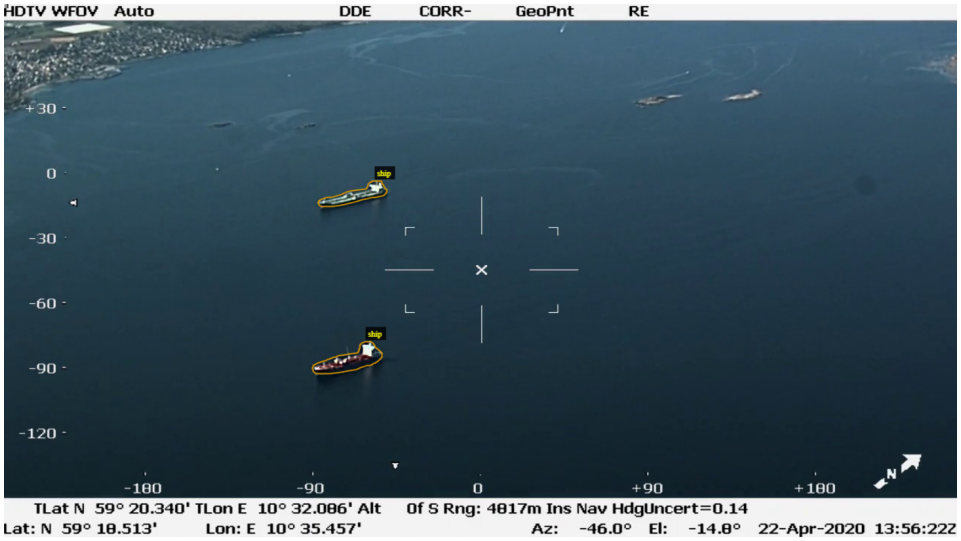


Figure 4.4: A sample image where the automatic annotation script has been applied. The pretrained Mask-RCNN model correctly detects two ships with satisfactory outlines

The resulting annotations can only serve as an *initial* annotation, as there has to be some human intervention to verify that the masks are of sufficient quality to train other networks, and remove the false positives. After running the automatic annotation script on the original dataset, one has to manually check each annotation and determine their quality and change them if necessary. In addition, one would be required to manually label the false negatives in the images. As such, the primary goal of the automatic annotation is only to *reduce* the amount of manual annotation as possible, rather than annotating the entire dataset. Paradoxically, the ideal annotation tool would achieve this, however if one had access to a model which was capable of this, it would likely make the training of another model on the resulting dataset redundant.

4.2.3 Training, Validation and Testing Divide

The division of the dataset into three subsets, training, validation and test set, is a common approach in machine learning applications. It is crucial that these subsets are independent of each other, i.e. that none of the data entries exist in more than one set. The training set contains the data entries which is used as input to the network in the backpropagation algorithm. In other words, this set is responsible for the learning of the network, and the performance of the network is mostly dependent on the quality of this set. The test set is used to determine the final performance of the network after it has been fully trained. The validation set on the other hand is used to determine the performance of the model during the training process. Most often, the model is tested on the validation set in certain intervals of the training, and the loss on the validation set is stored for analytic purposes afterwards. The validation loss is compared with the loss on the training set, and one can determine whether the model is overfitted if the validation loss starts to rise when the

training loss continues to fall. This is vital information for determining for how long one should train the dataset. It also allows for *early stopping*, which stops the training process automatically when this criteria is fulfilled.

Cross-Validation

Typically, the split into the three subsets are roughly divided with the proportions 60 % of the total dataset being dedicated to the training set, 20 % validation and 20 % test set. A problem which can arise when dividing the dataset in a trivial manner is *selection bias*. Some of the data may be more useful for the training process, and other for testing. For instance, if only images from the dataset containing one class or scenario were used in the training set, whereas the images from another scenario is strictly reserved for either the validation or testing, it is likely that the generated model will perform worse than if the training set was more balanced. The challenge is to determine which images should be selected for each set, and the balance between them.

There are multiple strategies for deciding on the division. Cross-validation is a term coined for multiple strategies that determine the optimal training-validation split. The idea behind cross-validation is to find the optimal division between validaiton and training set to avoid the problem of selection bias. There are numerous ways to perform cross validation, including holdout, K-fold and Repeated random subsampling. The simplest one to implement is Holdout, where one simply does not utilize a validation set. Thus, one skips a large part of the selection bias as all of the images are now included in the training set. While this is a clear advantage for smaller datasets where each image is proportionally more important than in a larger dataset, the down side is that one loses the bonuses of the validation set. The main drawback is not being able to detect if the training becomes overfitted, and early stopping is ruled out. Essentially, the training process becomes blind, and one can only conclude whether the network has become overfitted by after each training sequence and comparing the evaluation results. If the test results decrease when trained for more iterations, it is likely that it has become overfitted and the ideal training length is lower than the current one.

K-fold and Repeated random subsampling are more complex algorithms to determine the performance of the network on the dataset as a whole. Rather than sticking to one test set and one training set, they both are based upon selecting different fragments of the total dataset as each subset. K-fold select k different splits of the dataset, which results in k different training and test sets. The algorithm then trains the network for each training set, and evalutates its performance on the test set. After k iterations, all of the performance scores are averaged, and the result is an overall representation of how well the model performs across scenarios of the entire dataset.

Repeated random subsampling works in a similar fassion of splitting up the dataset in k fragments. However, the selection process towards each subset is random, and while using K-fold one is guaranteed to use every data entry in both training and testing over the course of the algorithm, the latter algorithm provides no such guarantee. Thus, there is a risk that

certain scenarios may never be tested.

The drawback of using the latter types of cross-validation is that they require several training and testing iterations to determine the overall performance. While they reduce both the chance of overfitting and the effect of selection bias, similar effects can be achieved by using the Holdout approach with a carefully chosen test set when the total dataset is fairly balanced.

4.3 Mask Annotated Datasets

The tables 4.1 and 4.2 show the numbers of annotated images and objects. The former table depicts the images from the specialization project dataset which have been annotated with masks rather than bounding boxes, and the latter table shows the mask-annotated images that has been extracted from the videos that were acquired during the master thesis.

Table 4.1: Overview of the Dataset from the Specialization Project. The training set is annotated with masks, whereas the Unrefined Test set is annotated with bounding boxes

Data Set	Number Of Images	Number of Objects
Training Set	3,703	4,272
Unrefined Test Set	1,516	2,448
Total Dataset	5,378	6,894

Table 4.2: Overview of the Additional Images

Data Set	Number Of Images	Number of Objects
Training Set Addition	238	421
Shoreline Test Set	204	266
All New Images	442	687

Final Datasets

Table 4.3: Overview of the final dataset, and the respective subsets. The training and shoreline set are annotated with masks, whereas the Unrefined Test set is annotated with bounding boxes.

Data Set	Number Of Images	Number of Objects
Training Set	3,941	4,693
Unrefined Test Set	1,516	2,448
Shoreline Test Set	204	266
Total Dataset	5,820	7,581

4.4 Summary of Dataset Creation Pipeline

- Capture videos or images containing the desired classes.
- Create a set of suitable images. This is a manual process where each image should be chosen based on its quality. In future works, this process could likely be automated with active learning.
- Annotate the images. First, run initial automatic annotation to reduce the workload. Subsequently, verify the quality of the automatic annotations, and annotate the missing objects. The initial automatic annotation scripts contains the possibility of both annotating bounding boxes or masks depending on the chosen network architecture.
- Split the dataset into training, validation and test sets.

Chapter 5

Training and Testing

This chapter will discuss the training and testing methodologies used in this thesis. In particular, it will address the functionalities in the popular computer vision framework Detectron2, and how they were utilized to gain the results. The framework is a powerful tool to build object detection applications upon. However, it may be difficult to orientate what functionality is available straight away, and what one has to implement oneself. This chapter will serve as a explanation of the training and testing processes utilized in this project, as well as a description of how to use common deep learning functionalities such as data augmentation and validation sets in the Detectron2 framework.

5.1 Detectron2

5.1.1 Installation

The framework itself requires initially little set up. There is a fairly good tutorial notebook explaining the installation process as well as how to train the networks on custom datasets. One can either install the dependencies using this notebook, or install the following libraries from scratch. The first step is to create a new virtual environment as is customary when using deep learning applications. The reason for this is that different frameworks have different requirements in terms of version control, and typically deep learning applications are very finicky about the versions of their dependencies. To differentiate between different deep learning applications, it is necessary to install different versions of the dependencies in separate virtual environments. The list of the Detectron2 dependencies are listed below:

- MacOS or Linux
- Python $>= 3.6$
- torch $>= 1.4$

- torchvision
- cython
- numpy
- cv2
- cocoapi

After installing the dependencies in the virtual environment, the Detectron2 framework itself is installed by first cloning the repository from Github, and subsequently installed it with Python.

5.1.2 Dataset Format

The inherent annotation format utilized by Detectron2 is the COCO format. If the respective dataset is structured in this format, the dataloader in Detectron can immediately detect the annotations and rewrites them automically to the Detectron format. Otherwise, one has to rewrite the get_object_dict(img_dir) function in the tutorial notebook. Depending on the format of the custom annotations, it might be easier to convert the dataset to the COCO format. If one uses the VIA tool, it is fairly short work to rewrite the get_object_dict(img_dir) function

5.1.3 Data Augmentation

Data augmentation techniques are often used to either artificially add more data samples to the dataset, or alter certain samples of the dataset with the aim of generalizing the model. In some cases the dataset only contains images with a certain color scheme, objects in certain positions of the images, a certain orientation or only certain subclasses. In such cases, there is a possibility that the final trained model will be biased towards the scenarios in the majority of data samples in the training set. When the biased model is used to detect objects in other scenarios than is contained in its training set, it would likely perform badly due to the bias. For instance, if all the images in the training set are taken in daylight, the model will likely perform worse when being applied to images in twilight or in darker light. By applying augmentation techniques such as random lighting for instance, one artificially creates data samples containing new scenarios, which hopefully leads to a more generalized model. Note that there is an uncertainty as there is no guarantee of its benefits, however data augmentation often leads to better performance.

The Detectron2 contains support for multiple data augmentation techniques. The total list of implemented augmentation functions can be found within
$/detectron2/data/transforms/transform\_gen.py$.

A selection of the functions are listed below:

- RandomApply

- RandomBrightness
- RandomContrast
- RandomCrop
- RandomExtent
- RandomFlip
- RandomSaturation
- RandomLighting
- RandomRotation
- Resize
- ResizeShortestEdge

If one wishes to select which techniques to use in training, it requires a bit of rewriting of certain mapping functions to apply them. In order to make full use of these techniques one has to write a custom dataloader. The default dataloader can be found in $/detectron2/data/dataset\_mapper.py$, where the dataloader class itself is called DatasetMapper. Within the class, one has to include a list of the supported augmentations from transforms_gen.py one wishes to perform. The custom dataloader has to included in the Trainer class.

More augmentations are not necessarily better performance wise. Instead of generalizing the data, at some point they introduce more noise, which is why the techniques themselves should be chosen carefully. In this project, RandomFlip was utilized as it was indicated that it increased performance in the specialization project, as well as ResizeShortestEdge to fit the shortest edge.

5.1.4 Validation Set

While other machine learning libraries like scikit-learn contain functionality for automatically splitting the training set into a training and validation split, the Detectron2 library contains no such functionality. If one desires to utilize a validation set to determine the object detection performance and validation loss during training, one has to implement this functionality oneself. There are multiple sources and blogs online describing the way to implement this. The general way to implement validation sets is manually splitting the dataset into training, validation and test sets, and subsequently register each subset. In addition, one has to augment the DefaultTrainer class in $/detectron2/engine/defaults.py$, to evaluate for each EVAL_PERIOD, which is a parameter the user defines in the config file.

5.1.5 Transfer Learning

Detectron2 contains a substantial *model zoo*, which essentially is a collection of pretrained models and networks that can be used to accelerate and boost the training process. It is customary when building a custom model on a smaller dataset to use transfer learning to reduce both the amount of data and time required to train an effective model. The available models for instance segmentation are listed in table 5.1.

Table 5.1: A list of the instance segmentation models available in Detectron2

Network	Backbone
Mask-RCNN	ResNet50 + C4
Mask-RCNN	ResNet50 + DC5
Mask-RCNN	ResNet50 + FPN
Mask-RCNN	ResNet101 + C4
Mask-RCNN	ResNet101 + DC5
Mask-RCNN	ResNet101 + FPN
Mask-RCNN	ResNeXt101 + FPN
Cascade RCNN	ResNet50 + FPN
Cascade RCNN	ResNeXt152 + FPN

These are a subset of the available instance segmentation networks that are the most relevant for the thesis. The models are all trained with the 3x schedule, which is equivalent to 37 COCO epochs. There are three combinations of baselines for each backbone network, each representing a technique for extracting the feature map from the backbone. The FPN variations represent the standard Mask-RCNN network, where the feature extraction simply uses a Feature Pyramid Network, and uses fully connected neural networks for predicting boxes and masks. The C4 versions uses conv4 variations of ResNet with a conv5 head, while the DC5 versions uses conv5 ResNet backbone with dilations in conv5, which was introduced in Dai et al. (2017). The model zoo also contains models for other popular computer vision tasks, such as object detection, person keypoint detection and panoptic segmentation.

According to Detectron2s baseline performances (FAIR) the FPN and DC5 variations have the best performance, which is why they were chosen to be backbone format of the models in this thesis. Utilizing transfer learning in Detectron2 is fairly simple, and the instruction is part of the excellent tutorial notebook. Simply find the pretrained model which one desires to fine-tune in the model zoo, and specify it in the config yaml file. The training script will subsequently download the weights from the url contained in the config file automatically when the training is initiated. The number of models is limited to those in the model zoo, however they are sufficient for standard instance segmentation and object detection purposes.

There is a configuration parameter called cfg.MODEL.BACKBONE.FREEZE_AT for determining which stages to freeze of the pre-trained weights. It has three possible values, if

the parameter is set to ”0”, then none of the weights are frozen and the network will train from scratch. If it is set to ”1”, then the first stage of the architecture, i.e. the backbone network will be frozen, and if it set to ”2” both of the stages will be frozen. When training with the parameter set to ”1”, one achieves regular transfer learning or fine-tuning of the network.

5.1.6 Training From Scratch

In addition to setting the cfg.MODEL.BACKBONE.FREEZE_AT value to ”0”, one also need to perform certain changes in the network architecture as well. Primarily, the ”Frozen Batch Normalizaion” layer needs to be changed to ”Group Normalization” in accordance with the method in (He et al. (2018)). Detectron2 provides one network variation with this change already implemented, which is Mask-RCNN with ResNet50 and FPN as backbone.

5.1.7 Hyperparameters

A key part in any deep learning application is the tuning of the hyperparameters of the networks. Detectron2 keeps track of all its hyperparameters and individual network settings in a separate configuration file in the YAML format. The key advantage of the YAML format is that one can create a base config file containing the mutual settings for multiple networks, which network specific config files can ”inherit” from using the _BASE_ command. This reduces the amount of superfluous code, and makes generating custom config files easier. One ”tunes” the hyperparameters by altering the values within the config file. Some key hyperparameters include the number of training iterations, the number of images per batch, the learning rate and the Region of Interest (ROI) batch size per image. An iteration in the Detectron2 implementation is a *mini-batch* consisting of the number of images per batch for each GPU available. The max number of iterations determines the length of training session. The difference between the number of images per batch and the ROI batch size per image is that the former parameter is used for the training of the backbone, where as the latter is used for the batch size for training the heads of the network, i.e. how many regions is extracted from each image to train the network head.

5.2 Training

5.2.1 Available Hardware

The hardware that was available for this master thesis was a significant upgrade from the specialization project, which made the training of instance segmentation networks feasible. The specifications of the hardware was two ASUS RTX2080Ti Turbo GPUs, an Intel LGA1151 i9 - 9900K CPU and 64 GBs of RAM. Only a single GPU was used at a time, however this proved to be more than sufficient to both fine-tune the instance segmentation networks and train them from scratch. This is largely due to the power of the RTX2080Ti cards, which contains 11 GB of video memory as well as 4,352 CUDA cores. The CUDA

cores are optimized for parallel computations and are thus suitable for deep learning purposes.

5.2.2 Training Sets

The networks were trained primarily on two training sets. The first training set contains the 3,703 images from the specialization with 4,272 mask annotations, whereas the second training set is the next iteration of the first with an additional 238 images and 421 mask annotations. The purpose of training on both of these sets is to determine the performance boost of supplementing the training set with more images and annotations.

5.2.3 Training From Scratch and Transfer Learning

Training from scratch refers to the process of training all the layers in the network, including the backbone network. More specifically, it means that the weights are initialized randomly before training and all of the layers are augmented during training. The results from the specialization project indicated that YOLOv3-tiny trained from scratch outperforms larger variations of the network which is trained using transfer learning. These results are supported by He et al. (2018).

In this thesis, both of these techniques are utilized for training. However, a slightly modified approach is applied when training from scratch. Instead of initializing the weights randomly, they are instead initialized as the pre-trained weights. This is a hybrid approach, where one trains the entire network, but the network gains a ”head start” on finding the optimum.

Tuning

The models that were trained from scratch was trained for a longer period than the transfer learning models to allow for convergence. It became apparent that when initializing the weights randomly, the model required even longer time to converge. It was deduced that the transfer learning models overfitted after being trained for longer than 10,000 iterations as the test results significantly declined for a larger number of max iterations. When training from scratch with pre-trained weights, the model converged after 30,000 iterations, while the model trained with randomly initialized weights converged after 45,000 iterations.

The ROI batch size per image variable was tuned primarily after the size of the backbone network. It was found that for the ResNet50 backbone, a ROI_BATCH_SIZE value of 64 performed the best, whereas for the ResNext101 bacbone, a ROI_BATCH_SIZE value of 128 gave the best results. The remaining hyperparameters were set to the default within the Detectron2 framework.

5.3 Testing

The test sets were designed to be comparable with real life applications of the Seahunter system, in other words the test images were selected if they contained smaller objects or objects from afar, multiple objects or challenging lighting and weather circumstances. This careful selection was made such that the performance on the test sets would simulate the performance in tougher conditions of the Seahunter application.

Two primary test sets were utilized to determine the model performances. The first and largest test set is the same that was developed in the specialization project, which contains 1,516 images and 2,448 objects. This test set is designated to be a benchmark for the models in the specialization project and the master thesis as it contains the ”hardest” images of the dataset, and to allow for a qualitative comparison between their performances.

The second test set is specifically designed to determine the performance of the models in shoreline environments. This scenario is only represented in the larger test set in a small extent, and it became apparent while running inference using the trained models that they predicted hard false positives of several small isles in the images. Thus, it became necessary to have a metric for how the models performed in such environments. The second test set consists exclusively of images from the videos that were gathered during the course of the thesis, and thus the two test sets are independent of each other.

The testing methodology itself involved running inference using the inbuilt COCOEvaluation functionality within the Detectron2 framework. The framework supports multiple evalutation metrics, however the COCO format is standard for instance segmentation applications and was therefore used a baseline in this thesis. The COCO evaluation format includes the following parameters, AP, AP_{50}, AP_{75}, AP_S, AP_M and AP_L. AP_{50} and AP_{75} represent the average precision scores when a true positive is defined by the prediction matching the ground truth boxes with an IoU score of 0.50 and 0.75 respectively. Intuitively, the latter average precision score requires a bounding box which fits better to the ground truth, and thus a model with high AP_{50} but low AP_{75}, detects the objects, but the resulting bounding boxes are poor. The scores AP_S, AP_M and AP_L represent how well the model detects small, medium and large objects respectively. Finally, the main AP score is an average of multiple AP scores with an increasing IoU threshold, starting at 0.5 and increasing with 0.05 until it reaches 0.95.

Chapter 6

Results

The results will be presented in the following order. First, the performance boost of enlarging the training set will be determined by comparing the results of models trained on a smaller and a larger training set respectively. Second, the results of training using transfer learning and training from scratch will be compared. Third, a comparison between the results from the specialization project and the master thesis will follow. Fourth, a summary of the best models and their performance. The first, second and fourth sections will include the results on both of the available test sets, whereas the third section will only include the results on the larger test set.

6.1 Enlarging the Training Set

One can tell by judging the results in table (6.1) and (6.2) that increasing the training set with only 238 images have a substantial effect on the object detection results. Each model in the table is trained with the same hyperparameters as their respective counterpart, the only difference being the training set sizes.

Table 6.1: Performance on the Unrefined Test Set. The score is given as bounding box AP. All of these models have been trained using transfer learning.

Models	Train Set	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask-RCNN + ResNet50 + FPN	3,703	38.7	73.9	33.9	17.9	**44.3**	54.6
Mask-RCNN + ResNeXt101 + FPN	3,703	37.4	74.3	30.3	16.8	40.2	54.5
Cascade-RCNN + ResNet50 + FPN	3,703	38.4	73.1	33.9	18.2	43.4	55.3
Mask-RCNN + ResNet50 + FPN	3,941	41.3	81.8	34.7	25.5	42.4	55.0
Mask-RCNN + ResNeXt101 + FPN	3,941	**42.5**	**82.0**	**36.6**	**25.8**	43.1	**56.4**
Cascade-RCNN + ResNet50 + FPN	3,941	40.5	79.2	34.5	24.9	40.9	55.0

Table 6.2: Performance on the shoreline test set. The score is given as bounding box AP. All of these models have been trained using transfer learning.

Models	Train Set	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask-RCNN + ResNet50 + FPN	3,703	47.5	76.4	50.1	40.6	65.4	79.0
Mask-RCNN + ResNeXt101 + FPN	3,703	44.4	67.2	47.5	29.6	67.6	**79.1**
Cascade-RCNN + ResNet50 + FPN	3,703	47.8	73.3	54.5	38.1	68.6	71.6
Mask-RCNN + ResNet50 + FPN	3,941	50.5	**85.5**	51.4	**42.2**	63.7	75.6
Mask-RCNN + ResNeXt101 + FPN	3,941	51.3	83.2	54.6	42.0	65.7	78.8
Cascade-RCNN + ResNet50 + FPN	3,941	**51.4**	82.5	**55.5**	41.4	**69.0**	57.4

The models which are trained on the larger training set outperforms the models trained on the smaller training set by a good margin. The difference is most apparent in the AP_{50} category, where the difference between the two Mask-RCNN + ResNeXt101 models is 7.7 points. The clue to realizing the reason behind the increase lies in the AP_S score. The other categories AP_M and AP_L are fairly similar in all the models trained on the smaller and the larger training set. However, the AP_S is markedly better in the models trained on the larger training set. This improvement in performance is due to that the additional images contain mostly smaller objects, and thus it is not surprising that the average precision score in that category increases. However, it interesting to note the decrease of the AP_M scores with the additional training images. This is likely connected with the imbalance in the training set which is introduced with the overweight of smaller objects.

All of the following models in the next sections are trained on the larger training set due to the increase in performance.

6.2 Transfer Learning vs. Training From Scratch

Two primary techniques were utilized for training the networks, training from scratch and transfer learning. The tables (6.3) and (6.4) shows the qualitative comparison of the two methodologies on the Unrefined Test Set and the shoreline test set respectively. The ”TF” notation represents that the model is trained using transfer learning. The other models are fully trained on the custom dataset. The notation ”PW” represents the proposed method with the weights initialized as the pre-trained weights and ”RW” represents that the weights are initialized randomly.

Table 6.3: Performance on the Unrefined Test Set. The first two models are trained with transfer learning, whereas the next two are trained from scratch and are initialized with pre-trained and random weights. All of these models are trained on the largest training set with 3,941 images.

Models	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask-RCNN + ResNet50 + FPN + TF	41.3	81.8	34.7	25.5	42.4	55.0
Mask-RCNN + ResNeXt101 + FPN + TF	42.5	82.0	36.6	25.8	43.1	**56.4**
Cascade-RCNN + ResNet50 + FPN + TF	40.5	79.2	34.5	24.9	40.9	55.0
Mask-RCNN + ResNet50 + FPN + PW	**44.5**	**85.1**	**39.6**	**32.3**	**44.6**	56.1
Mask-RCNN + ResNet50 + FPN + RW	37.6	76.1	28.9	19.8	43.5	50.2

Table 6.4: Performance on the shoreline test set. The first two models are trained with transfer learning, whereas the next two are trained from scratch and are initialized with pre-trained and random weights. All of these models are trained on the largest training set with 3,941 images.

Models	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask-RCNN + ResNet50 + FPN + TF	50.5	**85.5**	51.4	**42.2**	63.7	75.6
Mask-RCNN + ResNeXt101 + FPN + TF	51.3	83.2	54.6	42.0	65.7	**78.8**
Cascade-RCNN + ResNet50 + FPN + TF	**51.4**	82.5	**55.5**	41.4	**69.0**	57.4
Mask-RCNN + ResNet50 + FPN + PW	50.7	83.5	50.5	41.1	65.7	71.5
Mask-RCNN + ResNet50 + FPN + RW	37.4	72.5	34.6	28.3	55.0	16.7

Interestingly, the results in table (6.3) indicates that the proposed method of initializing the weights as pre-trained weights and subsequently training all the layers, outperforms the best of the transfer learned models as well as the model with randomly initialized weights. The PW model gains the best AP scores in almost every category except the AP_L score, where the larger transfer learned model Mask-RCNN + ResNeXt101 model performs slightly better. Specifically, the PW model gains both higher AP_{50} and AP_{75} than the other variations, which supports the third observation of He et al. (2018) that says models trained from scratch predict masks and bounding boxes of higher quality than transfer learned models.

The results in table (6.4) are less conclusive than in table (6.3). The overall AP is similar for all the models except the model with randomly initialized weights. The two models that are trained from scratch are slightly more vulnerable to predicting hard false positives than their transfer learned counterparts, as can be seen in the figure 7.1, and thus they perform worse on the shoreline test set.

Both the fine-tuned model, Mask-RCNN + ResNext101, and the fully trained model, Mask-RCNN + Resnet50 + PW, struggle with clusters of objects, as can be seen in figures 6.1a and 6.1b. The latter model detects more objects in the bay area, but neither model is able to detect any of the moored vessels.

6.3 Comparison with The Specialization Project Results

The results in table (6.5) show unsurprisingly that the state of the art networks such as Mask-RCNN and Cascade-RCNN outperform the networks from the specialization project, even before the aforementioned improvements of increasing the training set and training from scratch were applied. The main performance boost over the specialization project is mainly due to utilizing and training deeper networks. Due to lack of powerful hardware it was not feasible to train the Matterport implementation of Mask-RCNN at the time, and the model that was used for inference was a COCO pre-trained model. The other main model from the project, YOLOv3-tiny, is a significantly smaller network which was trained from scratch, and performed almost equally well on the Unrefined test set as Matterports Mask-RCNN. The models from the thesis all perform over twice as well as the models from the specialization project.

It is worth mentioning that there are different Mask-RCNN implementations used in the specialization project and the master project, where the former implementation was made by Matterport, and the latter was made by Facebook AI Research group.

Table 6.5: A comparison between the performance of the best models from the specialization project and the master thesis on the Unrefined Test Set. The score is given in bounding box AP. The comparison is only made with the AP50 and AP75 scores as they are the only scores that are available for all of the models. The scores from the specialization project are also converted from a 0 to 1 scale, to a 0 to 100 scale.

Models	AP_{50}	AP_{75}
YOLOv3-tiny + Default Anchors + PW	38.5	8.6
Mask-RCNN + ResNet-101	38.4	13.1
Mask-RCNN + ResNet50 + FPN + TF	81.9	34.7
Mask-RCNN + ResNet50 + FPN + PW	**85.1**	**39.6**
Mask-RCNN + ResNet50 + FPN + RW	74.8	23.0

6.4 Summary

Table 6.6: The total performances on the Unrefined Test Set. All of these models are trained on the largest training set with 3,941 images.

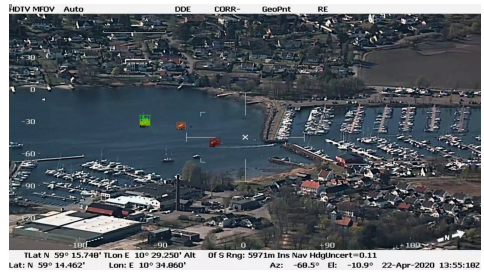
Models	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Cascade-RCNN + ResNet50 + FPN + TF	40.5	79.2	34.5	24.9	40.9	55.0
Mask-RCNN + ResNet50 + FPN + TF	41.3	81.8	34.7	25.5	42.4	55.0
Mask-RCNN + ResNeXt101 + FPN + TF	42.5	82.0	36.6	25.8	43.1	**56.4**
Mask-RCNN + ResNet50 + DC5 + TF	41.7	82.6	34.6	26.9	42.8	54.9
Mask-RCNN + ResNet101 + DC5 + TF	40.2	81.3	32.7	25.6	43.1	52.4
Mask-RCNN + ResNet50 + FPN + PW	**44.5**	**85.1**	**39.6**	**32.3**	**44.6**	56.1
Mask-RCNN + ResNet50 + FPN + RW	37.6	76.1	28.9	19.8	43.5	50.2

Table 6.7: The total performances on the shoreline test set. All of these models are trained on the largest training set with 3,941 images.

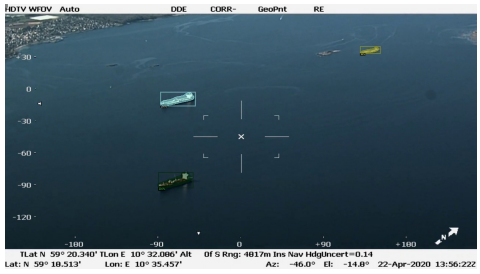
Models	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask-RCNN + ResNet50 + FPN + TF	50.5	**85.5**	51.4	**42.2**	63.7	75.6
Mask-RCNN + ResNeXt101 + FPN + TF	51.3	83.2	54.6	42.0	65.7	**78.8**
Cascade-RCNN + ResNet50 + FPN + TF	**51.4**	82.5	**55.5**	41.4	**69.0**	57.4
Mask-RCNN + ResNet50 + DC5 + TF	49.4	82.7	47.7	37.5	67.8	78.5
Mask-RCNN + ResNet101 + DC5 + TF	47.2	84.8	48.6	36.7	63.8	73.9
Mask-RCNN + ResNet50 + FPN + PW	50.7	83.5	50.5	41.1	65.7	71.5
Mask-RCNN + ResNet50 + FPN + RW	37.4	72.5	34.6	28.3	55.0	16.7

Overall, the best performing model on the Unrefined test set was trained from scratch and initialized with pre-trained weights on the larger training set. It especially outclassed the other variations in the AP_S category, where it gained a score of 32.3 and the closest variation received 25.8. In the smaller shoreline test set, it performed similarly to the transfer learned networks. The next best model was a transfer learned variation of Mask-RCNN with ResNeXt101 as backbone. It performed slightly better on the smaller test set. A selection of inference images of the two models are shown in figure (6.1). Both variations are able to detect very small objects, that models in the specialization project were not capable of. However, it appears that both models detect a significant amount of hard false positives of small isles.
Interestingly, Cascade-RCNN gained the best performance on the shoreline test set with the ResNeXt variation of Mask-RCNN next in line, with AP scores of 51.4 and 51.3 respectively.

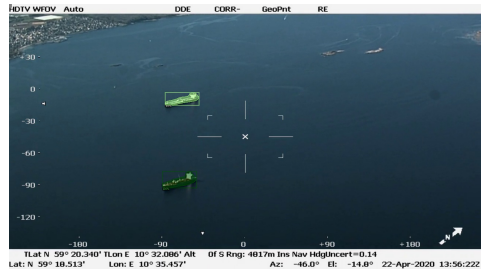
(a) Mask-RCNN + ResNext101 + TF



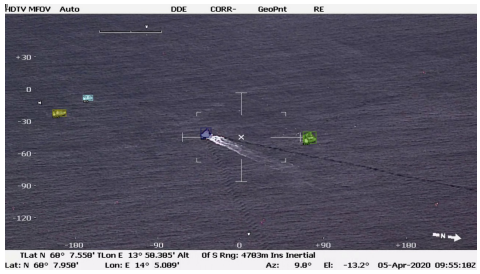
(b) Mask-RCNN + ResNet50 + PW



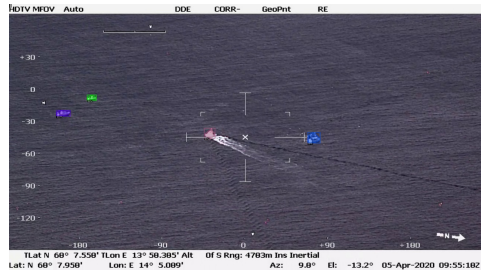
(c) Mask-RCNN + ResNext101 + TF



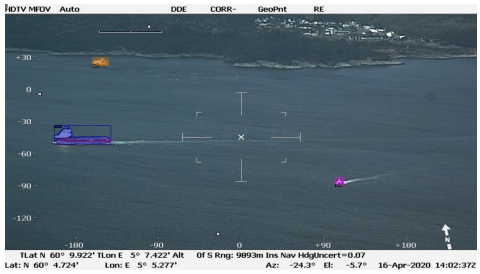
(d) Mask-RCNN + ResNet50 + PW



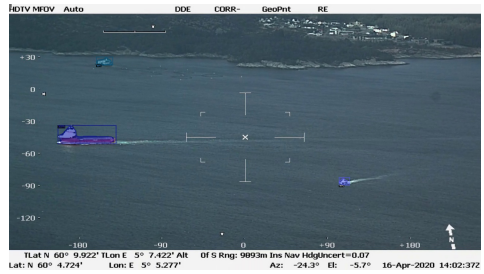
(e) Mask-RCNN + ResNext101 + TF



(f) Mask-RCNN + ResNet50 + PW



(g) Mask-RCNN + ResNext101 + TF



(h) Mask-RCNN + ResNet50 + PW

Figure 6.1: A selection of inference images from the two best models, Mask-RCNN with ResNeXt101 transfer learned, and Mask-RCNN with ResNet50 trained from scratch.

Chapter 7

Discussion

In this section, a selection of key areas in the thesis will be examined and discussed. First, a short review of the backbones will be given. Second, the problem of hard false positives is mentioned. Third, a discussion regarding the results of transfer learning follows. Fourth, an evaluation of the proposed dataset creation pipeline and possible alternatives. Lastly, a discussion whether the robustness of the models has increased from the specialization project to the models proposed in this thesis.

7.1 Backbone Performance

The different backbone architectures performed similarly on both test sets when trained with transfer learning. Despite consisting of twice as many layers, the ResNeXt101 variations did not significantly outperform the ResNet50 variations. In fact, in some categories it was even surpassed by its smaller version. This is likely due to that these variations were only trained on the layers after the backbone. The backbones themselves were trained on ImageNet, and thus it possible that the similar results of the different variations merely indicate the feature similarity between the custom dataset and ImageNet. Naturally, if the features within the custom dataset differ too much from the features within ImageNet then it is likely that the performance of the pre-trained backbone would saturate despite the increase in depth.

7.2 Hard False Positives

After training multiple models using both transfer learning and training from scratch, it is apparent that both methodologies struggle with hard false positives, which are faulty predictions with high confidence scores. However, it appears that the Cascade-RCNN architecture is slightly more resistant to this issue than Mask-RCNN. As seen in figure 7.1, the Mask-RCNN models detect multiple small isles as ships. The model trained from

scratch appears to detect more than its transfer learned counterpart. These models also detect incredibly small objects, however it seems that the models are not critical enough in their selection. Cascade-RCNN does not detect any of the isles in the image as ships. The reason the latter architecture performed well on the shoreline test is likely due to the relatively small number of HFPs it detects. It is probable that the refinement aspect of the cascade has a suppressing effect on these erroneous detections.

However, Cascade-RCNN is not immune to these errors. In figure 7.2 there is a selection of hard false positives among the true positives that are generated when running inference with Cascade-RCNN and the fully trained Mask-RCNN. Cascade-RCNN detects buoys and piers as ships.

There are possible solutions to further reduce the number of HFPs. The most obvious one is to increase the training set with more samples from this scenario. The overweight of images in the training set consists of ships at sea, and thus the trained model is likely biased towards this scenario and thus predicts the isles as ships. Supplementing the training set with even more images along the coast will likely passively suppress the numbers of hard false positives.

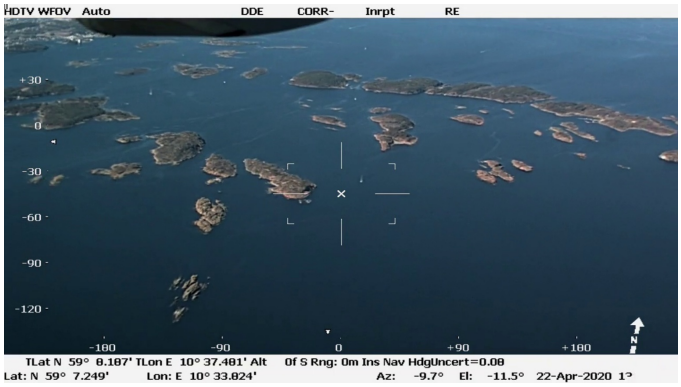
Another possible solution might be to fully train Cascade-RCNN. The refining nature of the cascade in combination with a backbone and head trained on a custom dataset, could properly learn the features within the dataset and be more critical in its predictions.

A more sophisticated approach is proposed in Cheng et al. (2018) where they actively suppress hard false positives by adding a separate classifier to the network. The second classifier is trained independently of the main two stage object detector, and is used to ”refine” the classifications from the base detector. While they utilized an older object detection two stage detector, Faster-RCNN, it should in theory be applicable to Mask-RCNN as well with some modifications to the loss function. Unfortunately, this method was not tested in this thesis due to time constraints.

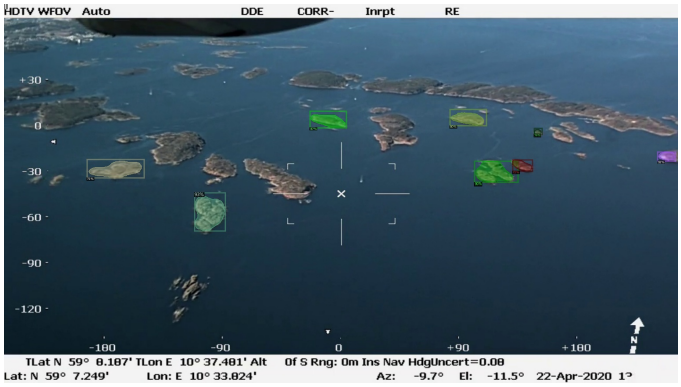
7.3 Transfer Learning vs. Training from Scratch

The results challenge the existing paradigm within computer vision where the normal training methodology involves using pre-trained weights and fine-tuning the network head. Both of the two training regimes performed well on the test sets. While the default method of randomly initializing the weights and training from scratch performed well, it did not converge to the levels of performance of the transfer learned models. Thus, it was a surprise that training from scratch with pre-trained weights performed better than both of the aforementioned models.

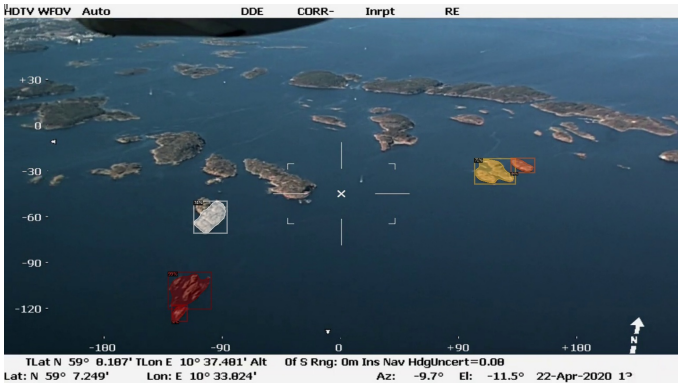
The results in He et al. (2018) suggests that models trained from scratch is both a viable and even effective training strategy, even when utilizing smaller datasets. They show that when utilizing smaller datasets of 35k to 10k images to train models from scratch the resulting models are as accurate as their pre-trained counterparts. However, this pattern



(a) Cascade-RCNN + ResNet50 + TF

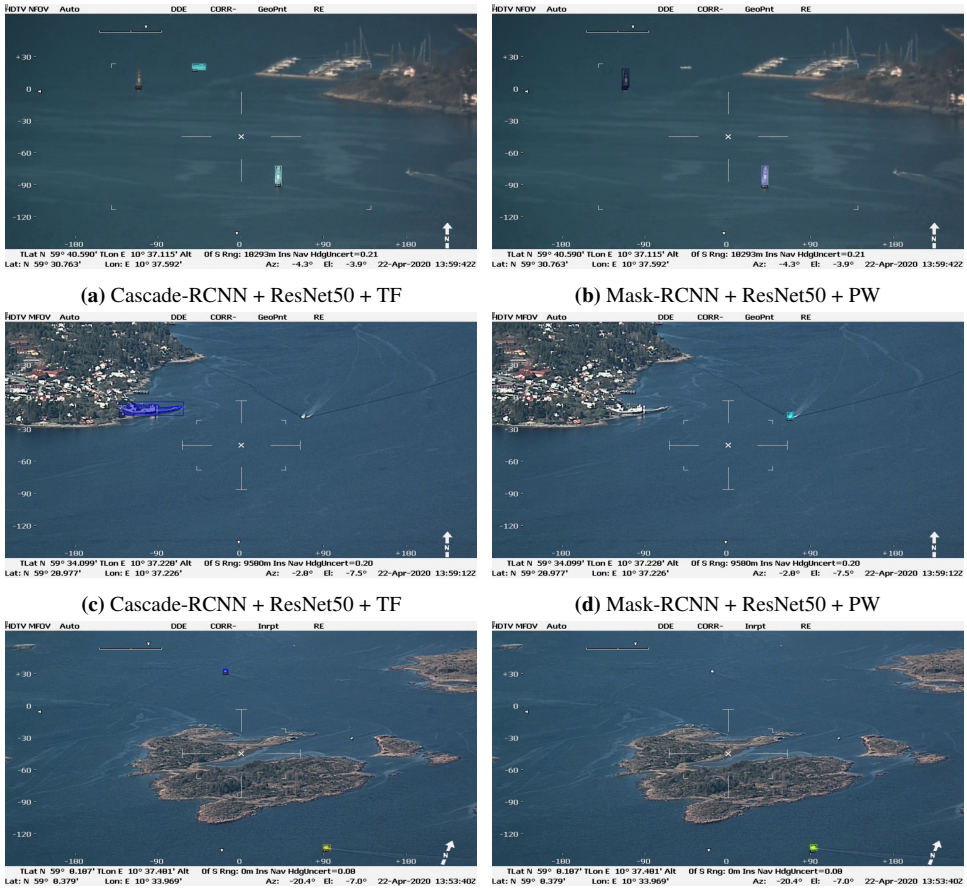


(b) Mask-RCNN + ResNet50 + PW



(c) Mask-RCNN + ResNeXt101 + TF

Figure 7.1: A selection of inference images with hard false positives. The images are generated with the two best performing models, Mask-RCNN + ResNext101 + TF and Mask-RCNN + ResNet50 + PW



(a) Cascade-RCNN + ResNet50 + TF

(b) Mask-RCNN + ResNet50 + PW

(c) Cascade-RCNN + ResNet50 + TF

(d) Mask-RCNN + ResNet50 + PW

(e) Cascade-RCNN + ResNet50 + TF

(f) Mask-RCNN + ResNet50 + PW

Figure 7.2: A selection of inference images with hard false positives. The images are generated with the two best performing models, Mask-RCNN + ResNext101 + TF and Mask-RCNN + ResNet50 + PW

does not continue, and when training on 1k images the model trained from scratch performs considerably worse. Thus, their experiments are inconclusive for models that are trained from scratch on training sets with less than 10k images but more than 1k. Astonishingly, the results from this thesis indicate that it is also viable to train deep networks such as Mask-RCNN with ResNet50 from scratch with only 4k images, and gain even better results than using a pre-trained backbone in some cases. It has to be noted that these results were gained by initializing the weights as the same pre-trained weights as their counterpart, and in that sense the model is not entirely trained ”from scratch”. The model initialized with random weights required far more training iterations to converge.

While the results of fully training the networks certainly are promising, the existing paradigm

of transfer learning is still incredibly useful in most deep learning applications. Transfer learning is the preferred method for applications with limitations in computing resources or lack of available data, as it speeds up the convergence of the model, and it achieves excellent results when applied in custom scenarios.

7.4 Evaluation of Deep Learning Pipeline

7.4.1 Initial Automatic Annotation Results

The essence of the automatic annotation is based on the inference of another pre-trained network, in this case Matterports implementation of Mask-RCNN. Despite the similarity between automatic annotation and instance segmentation, one can not utilize the average precision metric to evaluate the performance of the automatic annotation. While certain predictions may have an IoU score over a certain threshold, the outline itself may be of insufficient quality to keep as an annotation. For instance, some predictions may include pieces of background or the crosshair within the mask, which is too imprecise to use as an annotation.

In other words, a true positive for object detection metrics, is not necessarily a true positive for the automatic annotation. The metric for automatic annotation is therefore stricter than regular object detection metrics.

The initial automatic annotation was run for the entire old dataset (5,378 images), as well as all of the images extracted from the collected videos (567 images). In the newer 442 images, 652 objects were automatically annotated, where 166 of which were kept in the final annotations. The number of final annotations was 687, and thus the initial automatic annotation had an annotation accuracy of 25.46%, whereas it reduced the manual annotation load by 24.16%. Unfortunately, there is no data for how well the script worked for the existing dataset, but it did annotate a substantial amount of vessels successfully.

7.4.2 Supplementing the Dataset

The default mindset when creating datasets is ”the more data the better”, and the same could easily be applied to strategies for supplementing the dataset. However, there is a tradeoff to carelessly adding images to the set. One has to take into account the balance within the existing set, and add images in an equal manner. An imbalance was introduced when supplementing the dataset with the new images in this thesis, as they contained mostly smaller objects. This reduced the performance on larger objects slightly, which could theoretically be avoided if an equal amount of images with larger objects were added. On the other hand, the performance overall and the detection of smaller vessels in particular improved. The trade-off in this case was favorable, but one should keep the drawbacks in mind when adding images to the dataset. Alternatively, one could utilize more data augmentation to counteract the surplus of one image feature, but as mentioned previously this is not guaranteed to solve the problem.

7.4.3 Choice of Annotation Tool

During the late stages of the thesis, it became apparent that there exists tools with similar functionality as the proposed pipeline. Cvat (Intel) contains functionality for automatic annotation and frame by frame video annotation as well. However, as the existing code-base and dataset from the specialization project was written with the VIA tool in mind, it was not feasible to swap annotation tools. Due to the extra functionality, the Cvat annotation tool is recommended for further work.

7.4.4 Summary

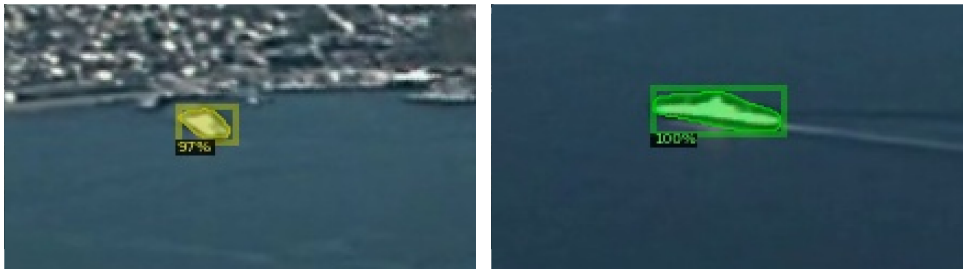
The dataset creation pipeline still requires a fair amount of human interaction and overview. The model used in initial annotation script can simply be swapped for a better performing model, which in turn would reduce the manual annotation load even further. However, the processes of gathering videos and selecting images are still very time consuming and are not trivial to automate. In addition, it is still required to manually check the quality of every annotation, and alter them if necessary.

The creation of datasets will likely always be the most time consuming in any supervised deep learning application. While utilizing weights pre-trained on existing large data collections significantly accelerates the process, a custom dataset will still have to made in most target scenarios. With that in mind, the proposed pipeline will hopefully aid in creating such datasets.

7.5 Improvement from the Specialization Project

One of the key improvements from specialization project is the improved ability to detect smaller objects. This is a vital functionality for usage in the practical application, as it is more useful to detect objects far away rather than objects in closer vicinity. This is in large part due to the ability to train large instance segmentation networks on the dataset. However, there are other attributors to the improvement as well. In the specialization project, part of the training set was subtracted to be used as a smaller training set, whereas in the thesis these images were reinstated to the training set. Thus, the training set was substantially larger in the latter project, and as shown in section 6.1 this may have had a positive impact on the models as well.

With the improvements of the thesis, it is necessary to ask whether these models now are robust enough to use in a real application. Despite the enormous leap in accuracy, the models still produce flaws such as detecting isles as ships. The performance boost of enlarging the dataset certainly motivates that the application in future might be robust enough for industrial use. However, it is likely that architectural changes are necessary to guarantee more consistent predictions. Overall, both the network architectures and the dataset requires refining before it can be used in an actual application.



(a) A small ferry. **(b)** A medium sized ferry.

Figure 7.4: Zoomed in versions of each detected object in figure (7.3)

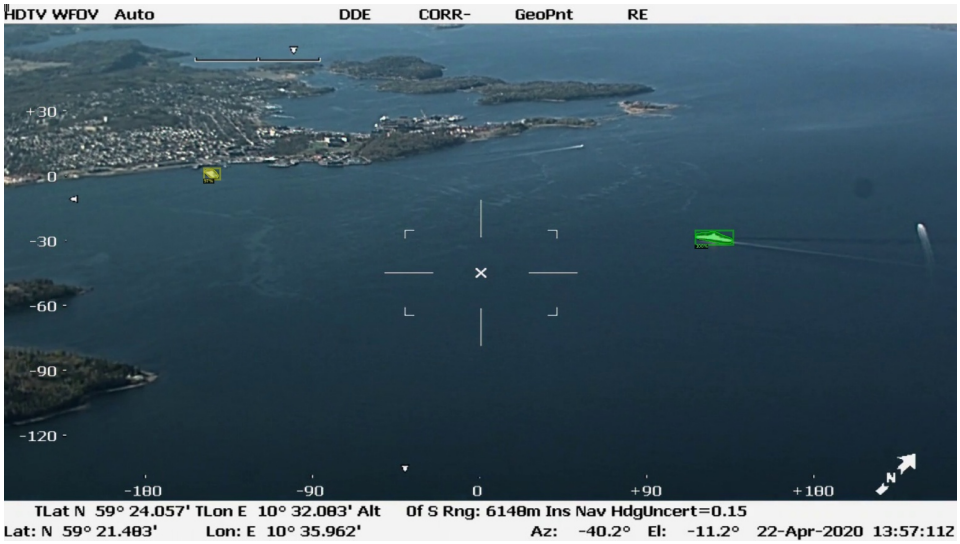


Figure 7.3: An inference image using Mask-RCNN with ResNet50 trained from scratch.

Chapter 8

Conclusion

In this thesis, two instance segmentation architectures, Mask-RCNN and Cascade-RCNN, were trained on a custom dataset of aerial images. Based on the results of the specialization project and He et al. (2018), the networks were trained from scratch and with transfer learning on the custom dataset to determine the respective merits of the individual training strategies. A hybrid strategy was also proposed, which involved initializing the weights as a set of pre-trained weights and subsequently training the network from scratch. On a test set of 1,516 images, the best performing model was generated by using the hybrid approach with Mask-RCNN with ResNet50 as backbone, which gained an AP score of 44.5. In comparison, a larger transfer learned model, Mask-RCNN with ResNeXT101 as backbone, gained an AP score of 42.5, and the model trained from scratch with randomly initialized weights gained a score of 37.6. On a smaller test set containing shoreline images, Cascade-RCNN gained the best performance with an AP score of 51.4. The results indicate that this architecture is slightly more resistant to *hard false positives*.

A dataset creation pipeline was proposed to accelerate the data gathering and annotation process. In particular, an *initial automatic annotation* process was made, using an existing pre-trained model for inference to create annotation proposals. This was used to annotate an existing dataset, and subsequently additional images with masks to allow for the training of state of the art instance segmentation networks with the aim of detecting naval vessels.

These results provide multiple insights. First, it is viable to fully train deep instance segmentation networks on smaller datasets down to 4,000 images. Second, the proposed method of training from scratch initialized with ImageNet pre-trained weights may outperform the methods of fully training with randomly initialized weights and training using a frozen, pre-trained backbone, when trained on a smaller custom dataset. Third, Cascade-RCNN and Mask-RCNN are highly suitable to detect naval vessels in aerial images, and increased the performance from the specialization project immensely.

The second insight supports the conclusion in the specialization project, where a smaller model trained from scratch may outperform a larger model trained with transfer learning, even when utilizing a smaller dataset. The results also support the observations in He et al. (2018), as fully training with randomly initialized weights did not perform as well as the fine-tuned model when trained on a dataset of less than 10k images. However, the proposed method of fully training the network and initializing the weights as pre-trained weights proved itself as a proper contender to these training regimes.

Chapter 9

Further Work

The performance increased significantly by increasing the training set, and strategies for gathering data and supplementing the dataset should be implemented in future. This task will always be prevalent in supervised computer vision tasks, and should be a focus point for further development. It was infeasible to implement multiple classes within the thesis due to insufficient data of individual classes. Additional footage will also allow for multiple subclasses such as ferries, cargo ships and recreational ships, but also entirely new classes within the dataset.

Other methods of improving the performance include applying more data augmentation. The newly published paper Zoph et al. (2020) suggests that stronger data augmentation considerably strengthen the performance of networks trained on smaller datasets, and thus additional augmentation techniques should be utilized in future training processes. This might also help with suppressing hard false positives, as more data is synthetically generated. However, this issue should also be actively dealt with. The results of adding a separate classification network in addition to a two-stage network in Cheng et al. (2018) are promising, and should be investigated in future as a measure to counteract these false predictions.

Real-time object detection on video should also be researched further. While the instance segmentation architectures utilized in this thesis are not suitable for this purpose, other architectures such as YOLACT (Bolya et al. (2019)) may be the solution. It performs real-time instance segmentation, albeit with a slight hit to the performance. It also requires considerably powerful hardware to run inference. However, if such conditions are met then it might also be worth to have a look at in further applications within this field.

Bibliography

Bolya, D., Zhou, C., Xiao, F., Lee, Y.J., 2019. Yolact++: Better real-time instance segmentation. arXiv:1912.06218.

Cheng, B., Wei, Y., Shi, H., Feris, R., Xiong, J., Huang, T., 2018. Decoupled classification refinement: Hard false positive suppression for object detection. arXiv:1810.04002.

COCOConsortium, . Coco: Common objects in context. http://cocodataset.org/home.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y., 2017. Deformable convolutional networks. arXiv:1703.06211.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T., 2013. Decaf: A deep convolutional activation feature for generic visual recognition. arXiv:1310.1531.

FAIR, . Detectron2. https://github.com/facebookresearch/detectron2.

Girshick, R., 2015. Fast r-cnn. arXiv:1504.08083.

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2013. Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524.

Grini, S., 2019. Object Detection in Maritime Environments. Master's thesis. NTNU.

He, K., Girshick, R., Dollár, P., 2018. Rethinking imagenet pre-training. arXiv:1811.08883.

He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask r-cnn. arXiv:1703.06870.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. arXiv .

Intel, . Cvat. https://github.com/opencv/cvat.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2016. Feature pyramid networks for object detection. arXiv:1612.03144.

Liu, Y., Wang, Y., Wang, S., Liang, T., Zhao, Q., Tang, Z., Ling, H., 2019. Cbnet: A novel composite backbone network architecture for object detection. arXiv .

Matterport, . Matterports mask-rcnn. https://github.com/matterport/Mask_RCNN.

Nie, S., Jiang, Z., Zhang, H., Cai, B., Yao, Y., 2018. Inshore ship detection based on mask-rcnn. ieee .

PapersWithCode, . Papers with code, instance segmentation on coco. https://paperswithcode.com/sota/instance-segmentation-on-coco.

Redmon, J., Farhadi, A., 2016. Yolo9000: Better, faster, stronger. arXiv .

Ren, S., He, K., Girshick, R., Sun, J., 2016. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv .

Rosenblatt, F.F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65 6, 386–408.

Shaodan, L., Chen, F., Zhide, C., 2019. A ship target location and mask generation algorithms based on mask rcnn. International Journal of Computational Intelligence Systems 12, 1134–1143. URL: https://doi.org/10.2991/ijcis.d.191008.001, doi:https://doi.org/10.2991/ijcis.d.191008.001.

Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition. arXiv .

StanfordVisionLab, . Imagenet. http://www.image-net.org/.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014. Going deeper with convolutions. arXiv:1409.4842.

VGG, . Vgg image annotator (via). http://www.robots.ox.ac.uk/~vgg/software/via/.

Vik, P., 2019. Detection of naval vessels using deep learning on aerial images.

Wu, Y., He, K., 2018. Group normalization. arXiv:1803.08494.

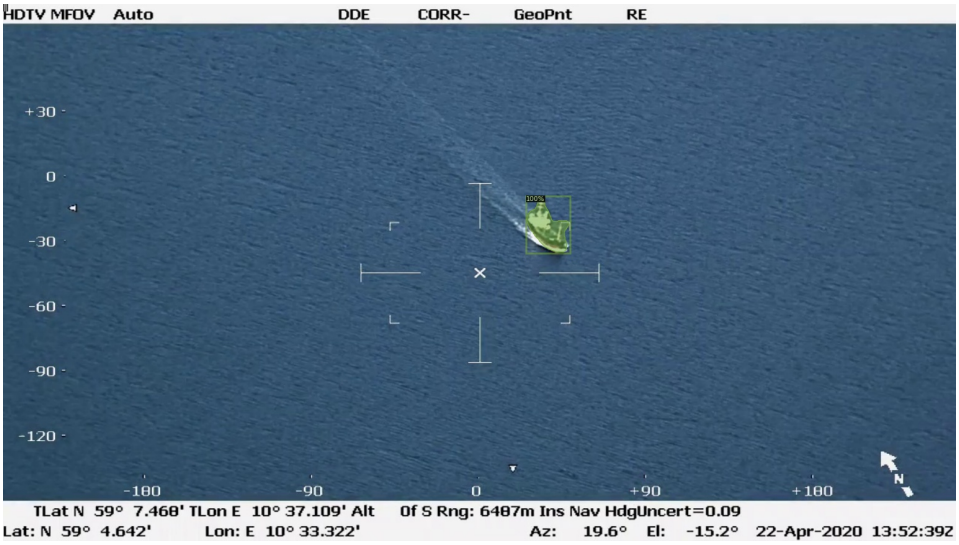
Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K., 2016. Aggregated residual transformations for deep neural networks. arXiv:1611.05431.

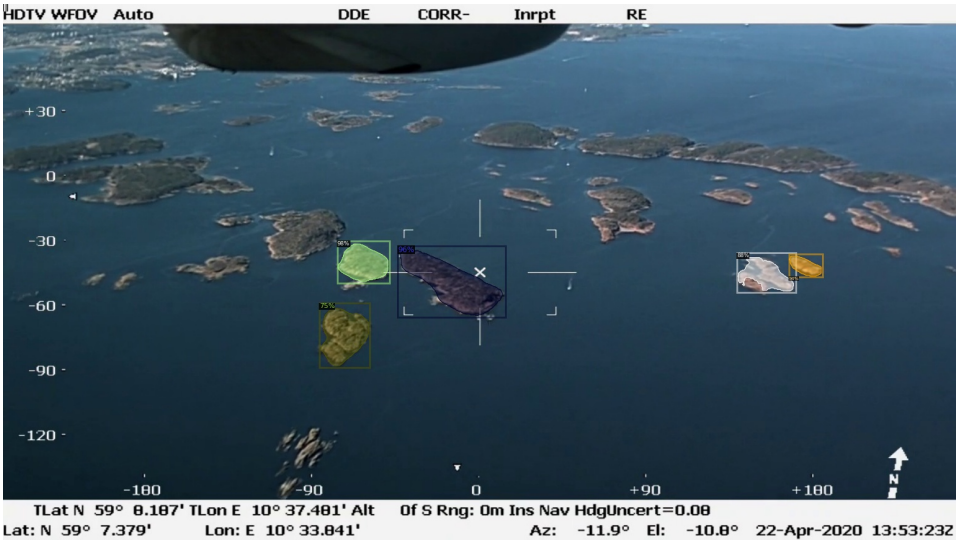
Zoph, B., Ghiasi, G., Lin, T.Y., Cui, Y., Liu, H., Cubuk, E.D., Le, Q.V., 2020. Rethinking pre-training and self-training. arXiv:2006.06882.

Appendix

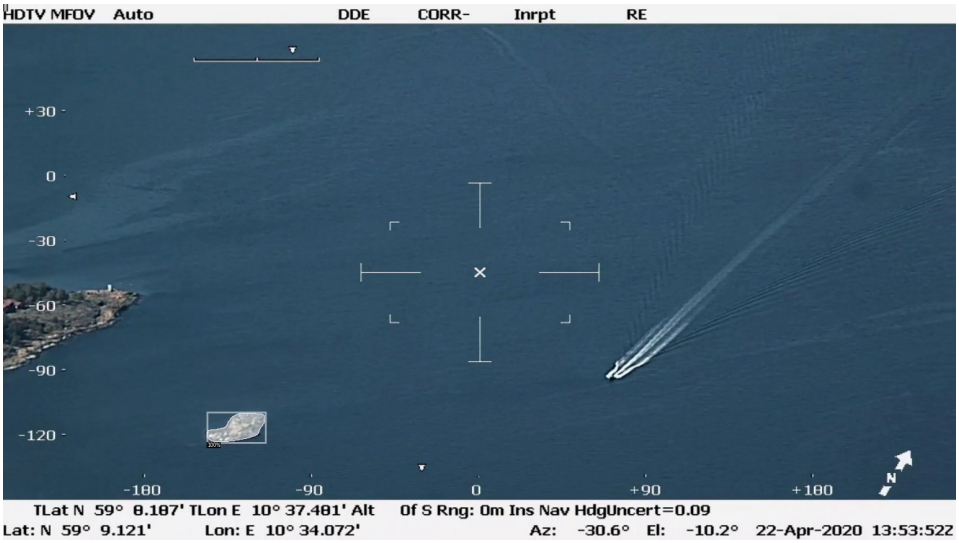
A.1 Additional Inference Images

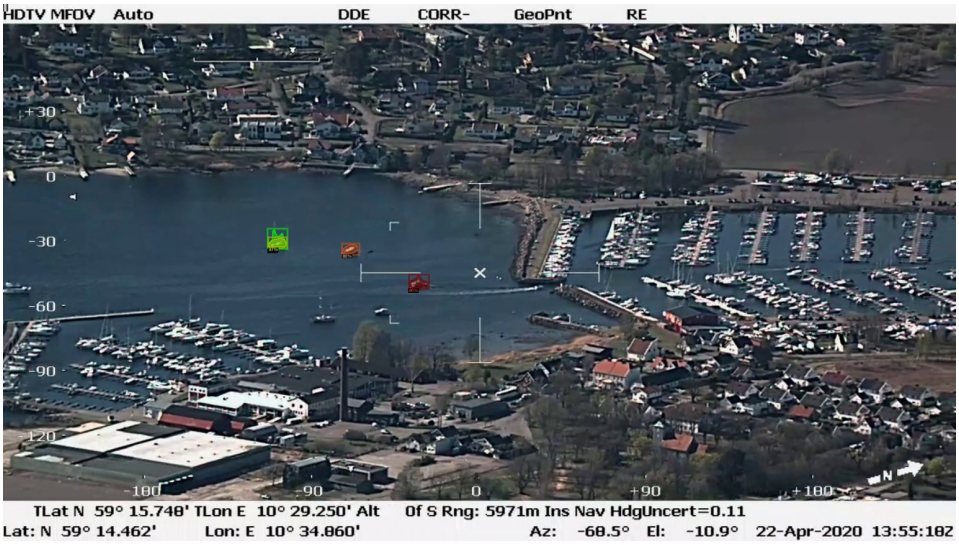
A.1.1 Mask-RCNN + ResNet50 + PW

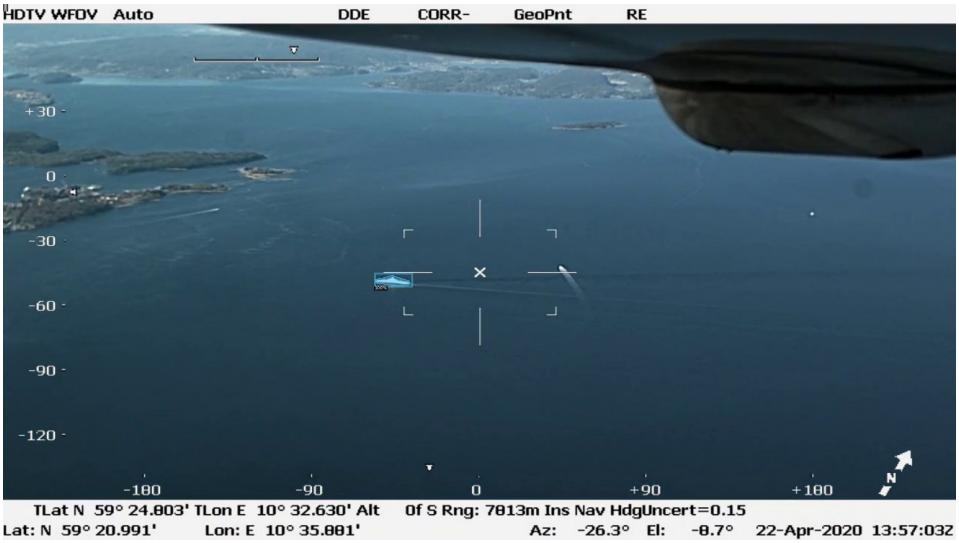


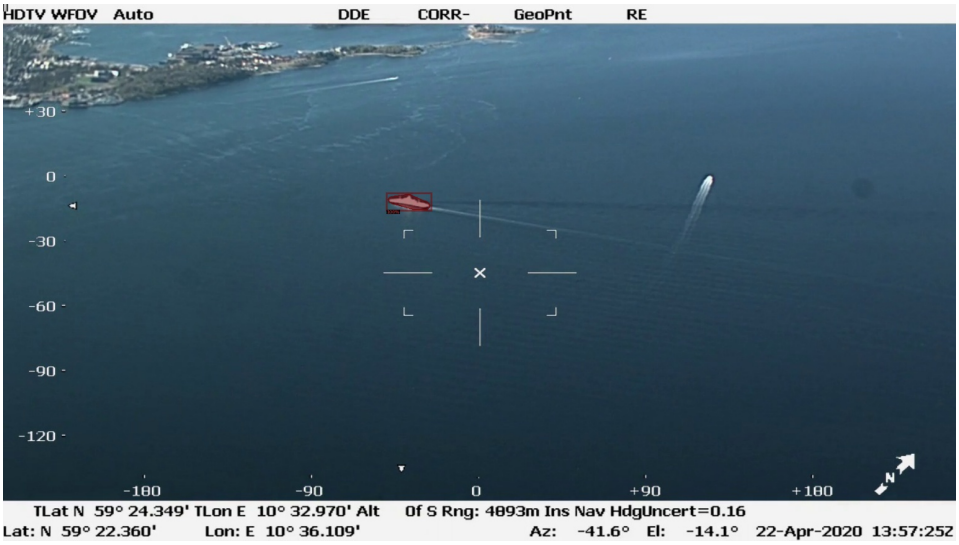

HDTV WFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 8.187' TLon E 10° 37.481' Alt Of S Rng: 0m Ins Nav HdgUncert=0.08
Lat: N 59° 7.379' Lon: E 10° 33.841' Az: -11.9° El: -10.8° 22-Apr-2020 13:53:23Z

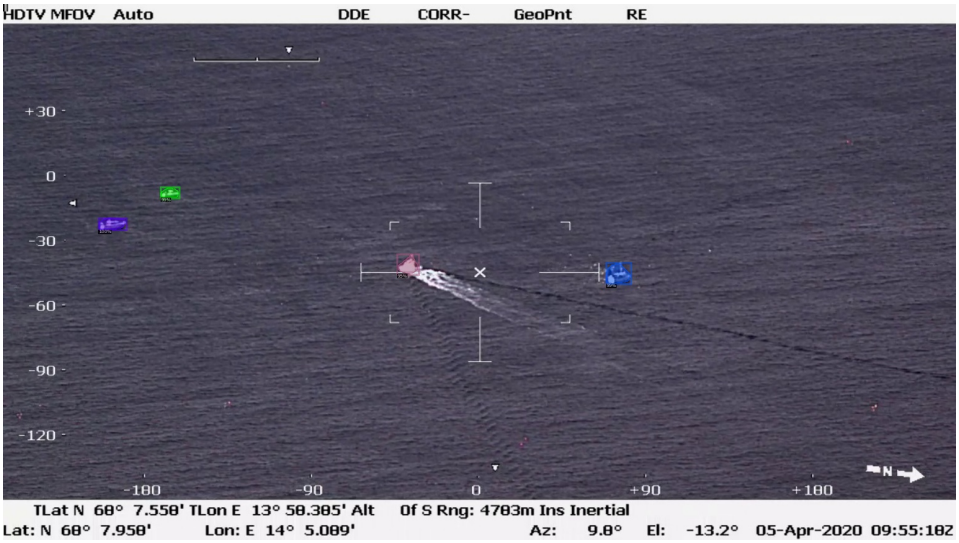

HDTV MFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 8.187' TLon E 10° 37.481' Alt Of S Rng: 0m Ins Nav HdgUncert=0.08
Lat: N 59° 8.379' Lon: E 10° 33.969' Az: -20.4° El: -7.0° 22-Apr-2020 13:53:40Z

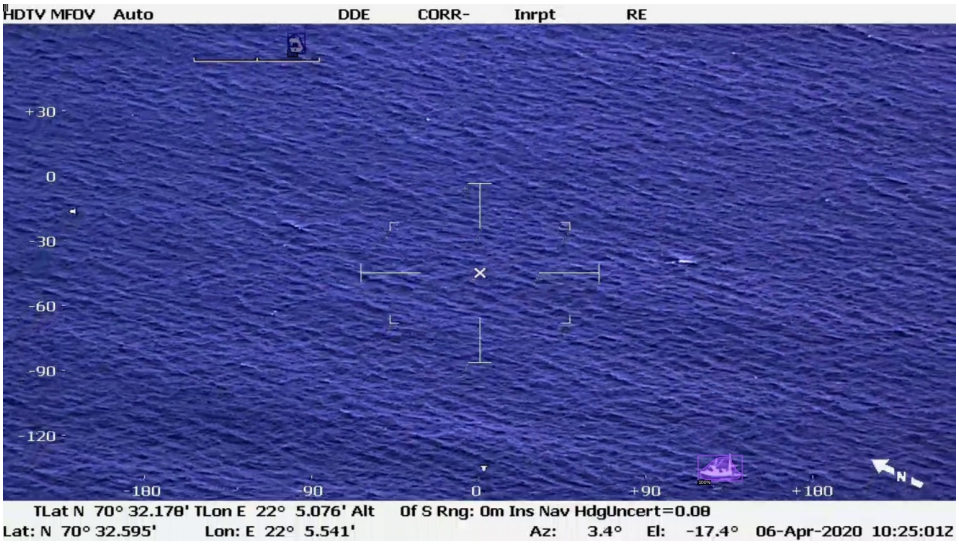

HDTV MFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180 -90 0 +90 +180
N
TLat N 59° 8.187' TLon E 10° 37.481' Alt Of S Rng: 0m Ins Nav HdgUncert=0.09
Lat: N 59° 9.121' Lon: E 10° 34.072' Az: -30.6° El: -10.2° 22-Apr-2020 13:53:522

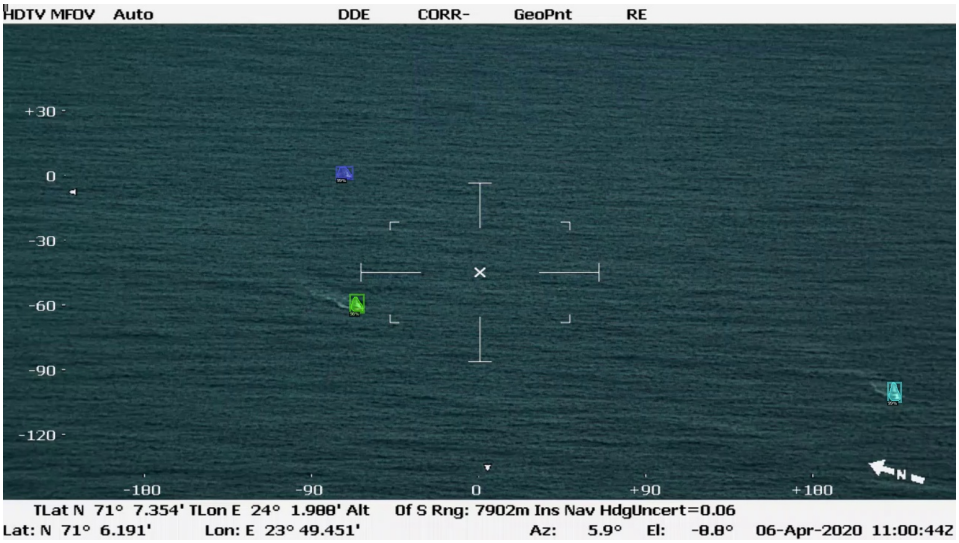

HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180 -90 0 +90 +180
N
TLat N 59° 15.748' TLon E 10° 29.250' Alt Of S Rng: 5971m Ins Nav HdgUncert=0.11
Lat: N 59° 14.462' Lon: E 10° 34.860' Az: -68.5° El: -10.9° 22-Apr-2020 13:55:182


HDTV WFOV Auto
DDE
CORR-
GeoPnt
RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 24.803' TLon E 10° 32.630' Alt
Of S Rng: 7813m Ins Nav HdgUncert=0.15
Lat: N 59° 20.991'
Lon: E 10° 35.881'
Az: -26.3°
El: -8.7°
22-Apr-2020 13:57:03Z

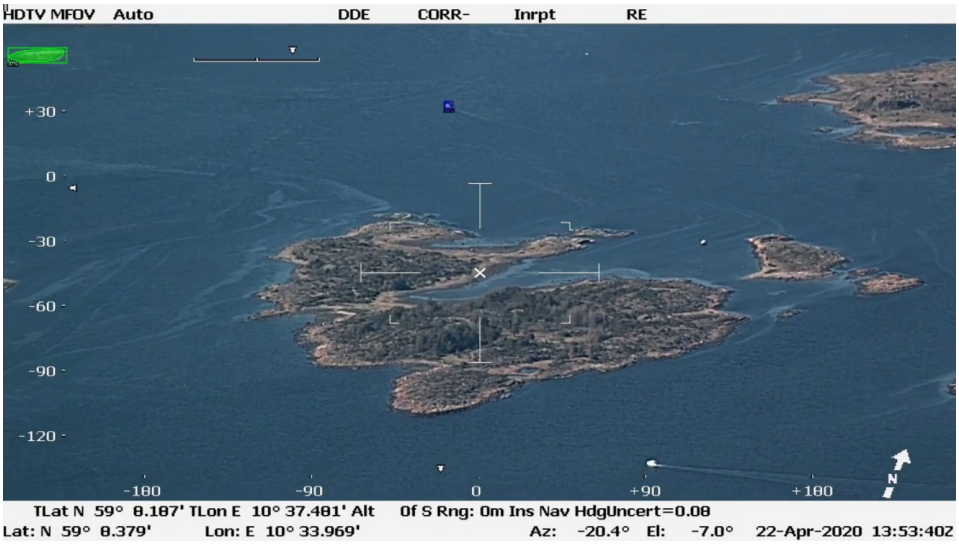

HDTV WFOV Auto
DDE
CORR-
GeoPnt
RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 24.349' TLon E 10° 32.970' Alt
Of S Rng: 4893m Ins Nav HdgUncert=0.16
Lat: N 59° 22.360'
Lon: E 10° 36.109'
Az: -41.6°
El: -14.1°
22-Apr-2020 13:57:25Z


HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 68° 7.558' TLon E 13° 58.385' Alt Of S Rng: 4783m Ins Inertial
Lat: N 68° 7.958' Lon: E 14° 5.089' Az: 9.8° El: -13.2° 05-Apr-2020 09:55:18Z


HDTV MFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 70° 32.178' TLon E 22° 5.076' Alt Of S Rng: 0m Ins Nav HdgUncert=0.08
Lat: N 70° 32.595' Lon: E 22° 5.541' Az: 3.4° El: -17.4° 06-Apr-2020 10:25:01Z

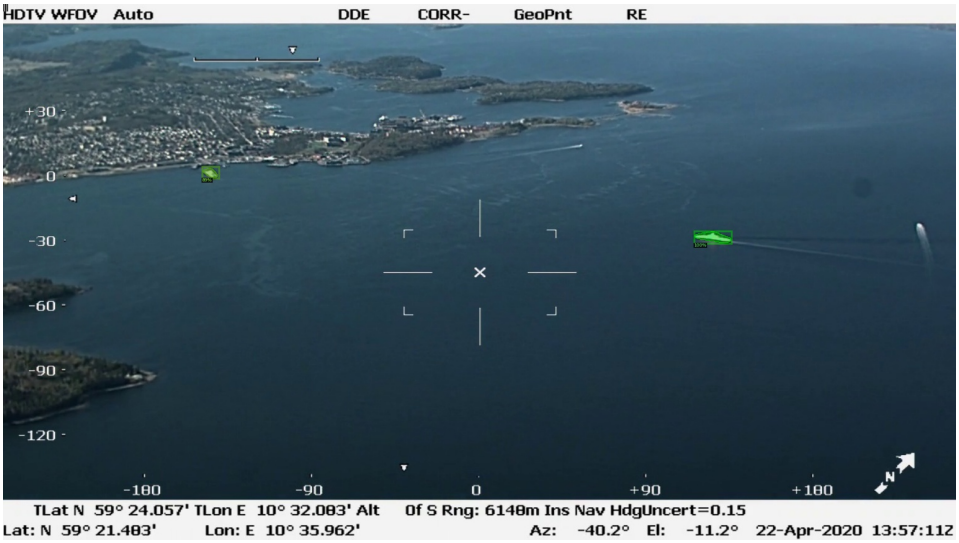


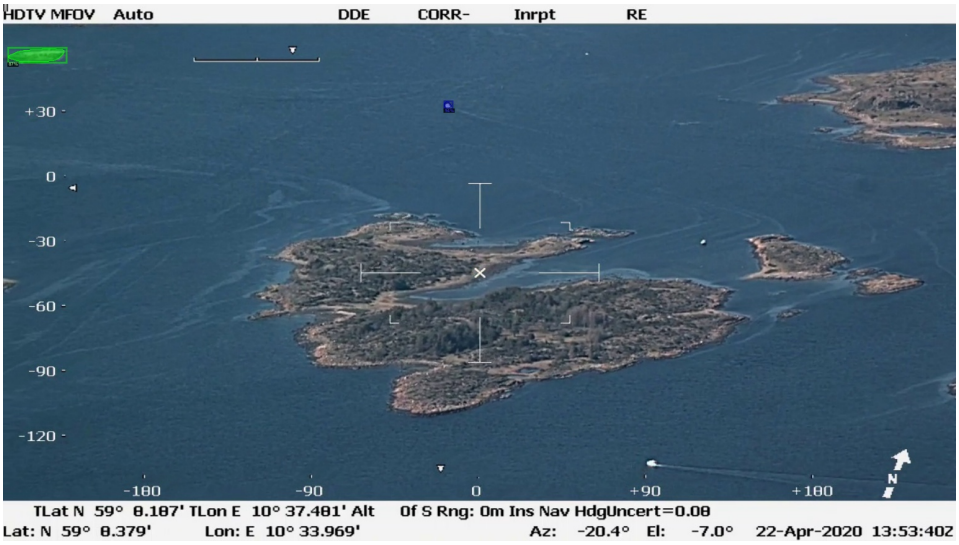
A.1.2 Mask-RCNN + ResNeXt101 + TF

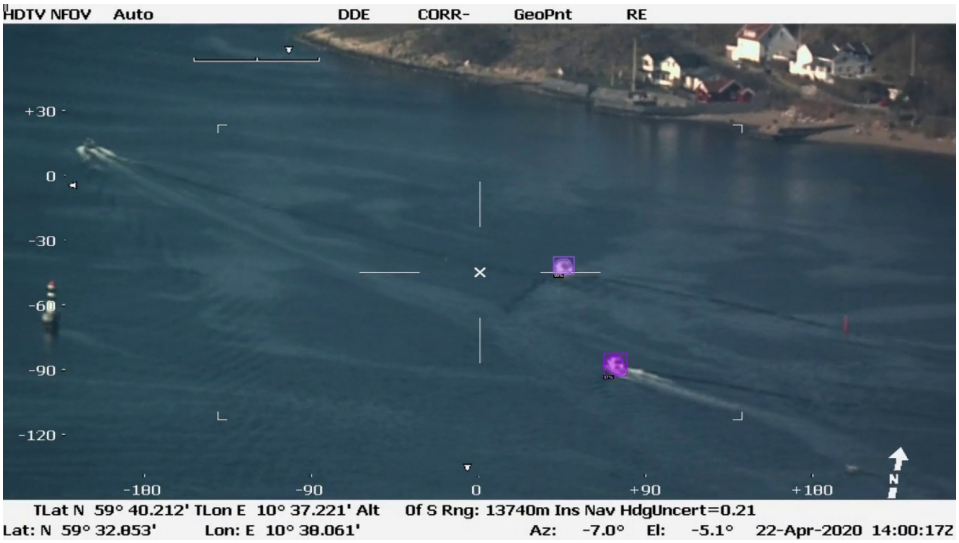


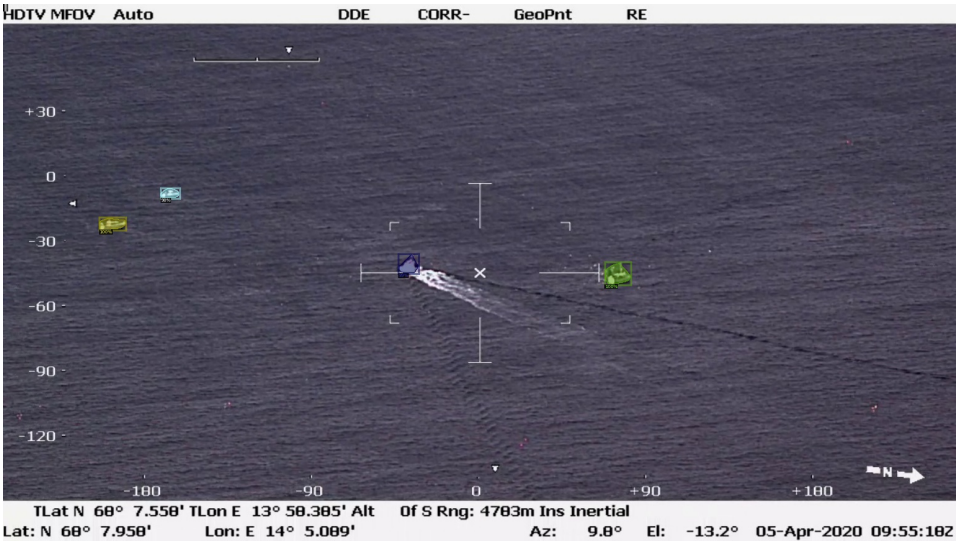

HDTV WFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180 -90 0 +90 +180
N
TLat N 59° 8.187' TLon E 10° 37.481' Alt Of S Rng: 0m Ins Nav HdgUncert=0.09
Lat: N 59° 9.868' Lon: E 10° 34.180' Az: -28.4° El: -8.2° 22-Apr-2020 13:54:03Z

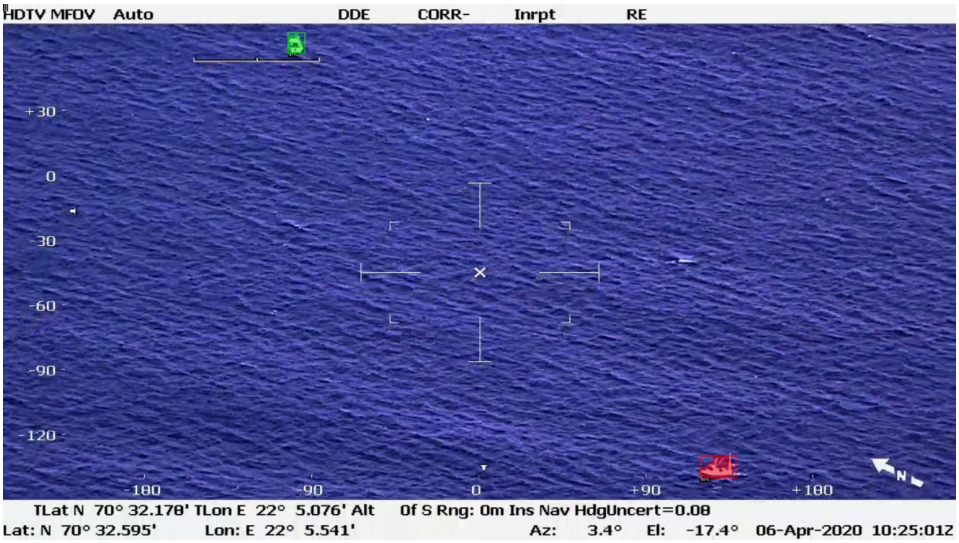

HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180 -90 0 +90 +180
N
TLat N 59° 15.748' TLon E 10° 29.250' Alt Of S Rng: 5971m Ins Nav HdgUncert=0.11
Lat: N 59° 14.462' Lon: E 10° 34.860' Az: -68.5° El: -10.9° 22-Apr-2020 13:55:18Z

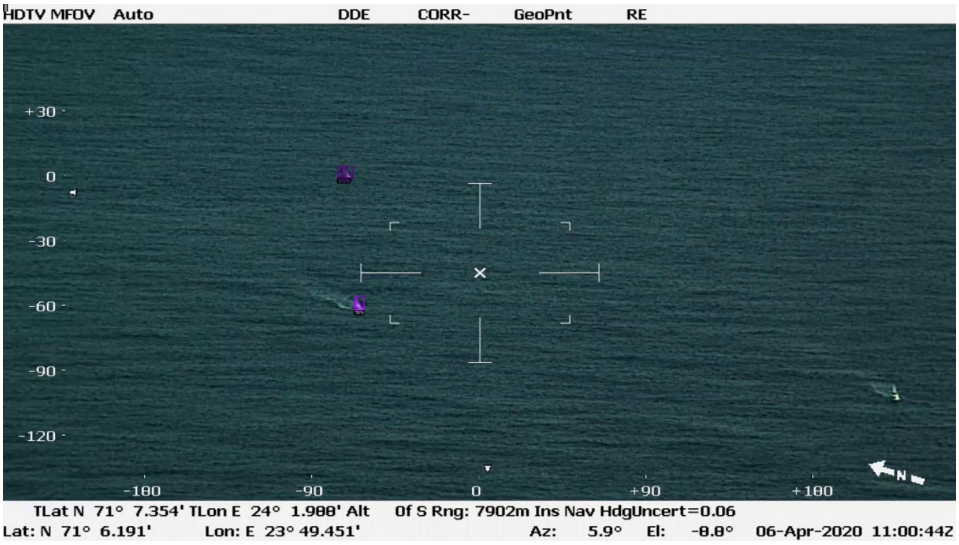

HDTV WFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 24.057' TLon E 10° 32.083' Alt Of S Rng: 6148m Ins Nav HdgUncert=0.15
Lat: N 59° 21.483' Lon: E 10° 35.962' Az: -40.2° El: -11.2° 22-Apr-2020 13:57:112


HDTV MFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 8.187' TLon E 10° 37.481' Alt Of S Rng: 0m Ins Nav HdgUncert=0.08
Lat: N 59° 8.379' Lon: E 10° 33.969' Az: -20.4° El: -7.0° 22-Apr-2020 13:53:402


HDTV NFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 40.212' TLon E 10° 37.221' Alt Of S Rng: 13740m Ins Nav HdgUncert=0.21
Lat: N 59° 32.853' Lon: E 10° 38.061' Az: -7.0° El: -5.1° 22-Apr-2020 14:00:172

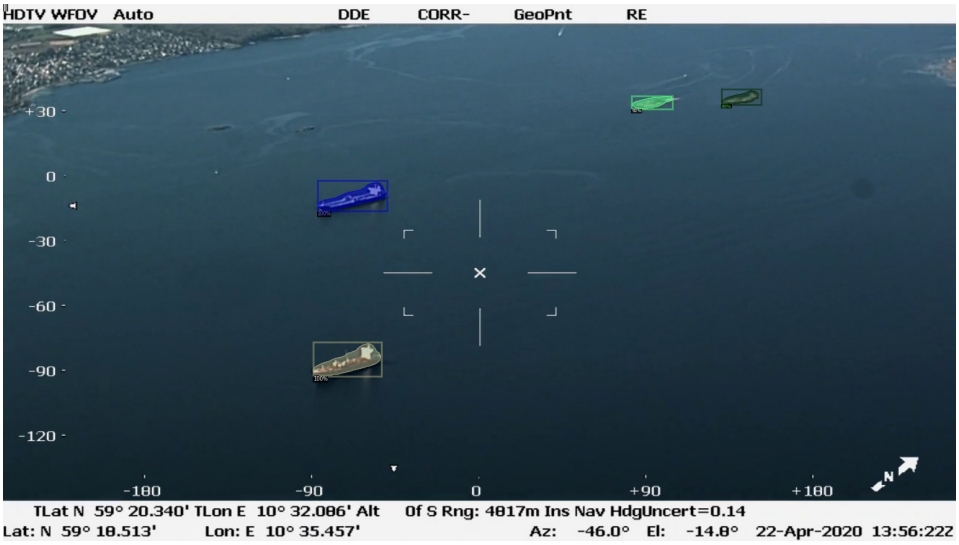

HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 68° 7.558' TLon E 13° 58.385' Alt Of S Rng: 4783m Ins Inertial
Lat: N 68° 7.958' Lon: E 14° 5.089' Az: 9.8° El: -13.2° 05-Apr-2020 09:55:182

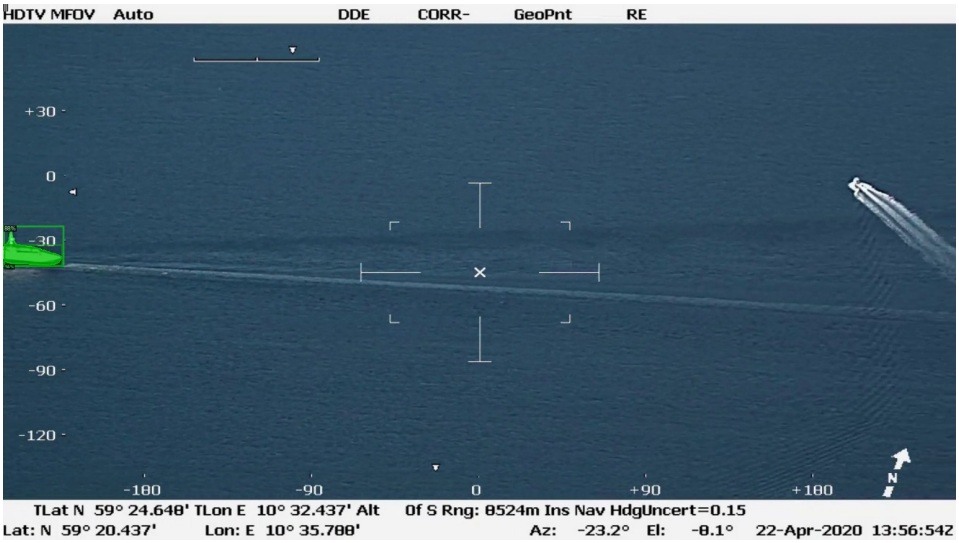

HDTV MFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 70° 32.178' TLon E 22° 5.076' Alt Of S Rng: 0m Ins Nav HdgUncert=0.08
Lat: N 70° 32.595' Lon: E 22° 5.541' Az: 3.4° El: -17.4° 06-Apr-2020 10:25:01Z

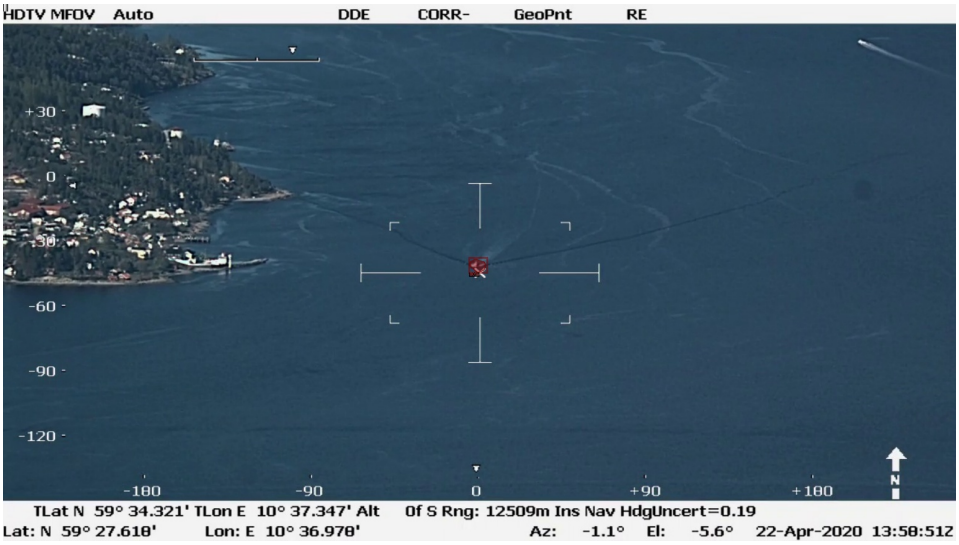

HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 71° 7.354' TLon E 24° 1.988' Alt Of S Rng: 7902m Ins Nav HdgUncert=0.06
Lat: N 71° 6.191' Lon: E 23° 49.451' Az: 5.9° El: -8.8° 06-Apr-2020 11:00:44Z

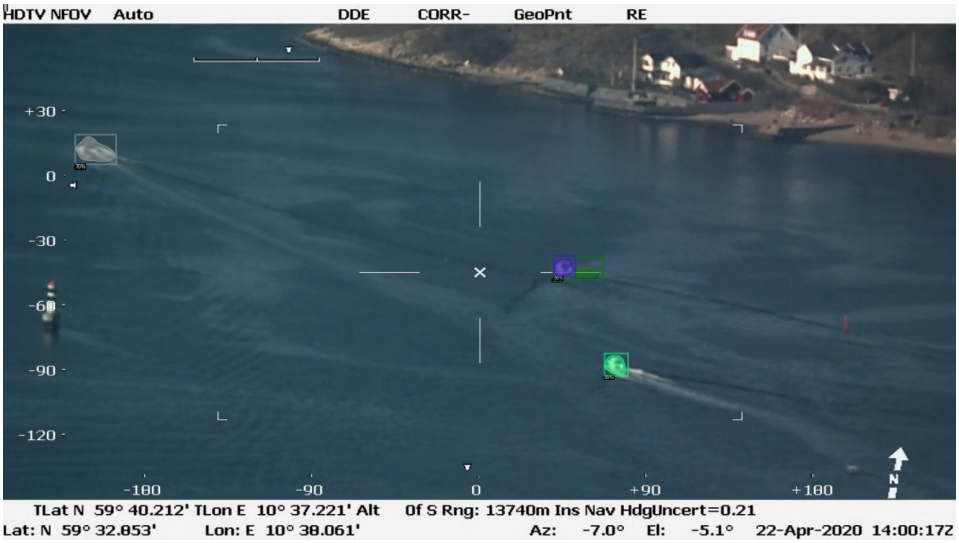
A.1.3 Cascade-RCNN + ResNet50 + TF

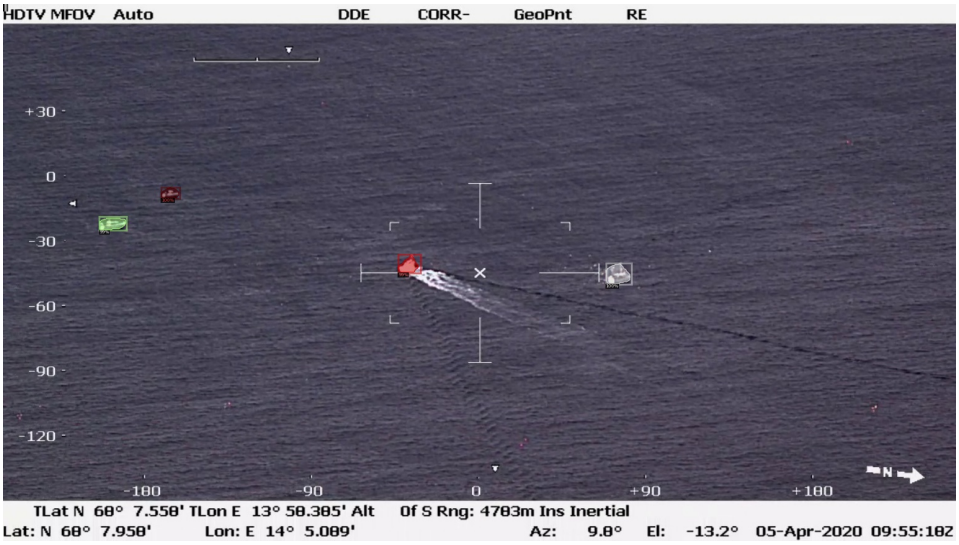

HDTV MFOV Auto
DDE
CORR-
GeoPnt
RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 15.748' TLon E 10° 29.250' Alt
Of S Rng: 5971m Ins Nav HdgUncert=0.11
Lat: N 59° 14.462'
Lon: E 10° 34.860'
Az: -68.5° El: -10.9°
22-Apr-2020 13:55:182

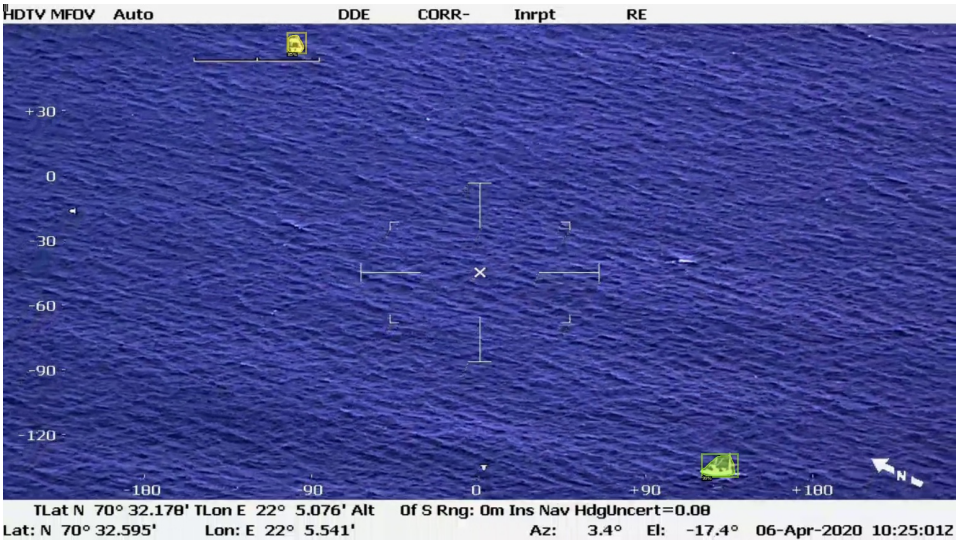

HDTV WFOV Auto
DDE
CORR-
GeoPnt
RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 20.340' TLon E 10° 32.086' Alt
Of S Rng: 4817m Ins Nav HdgUncert=0.14
Lat: N 59° 18.513'
Lon: E 10° 35.457'
Az: -46.0° El: -14.8°
22-Apr-2020 13:56:222

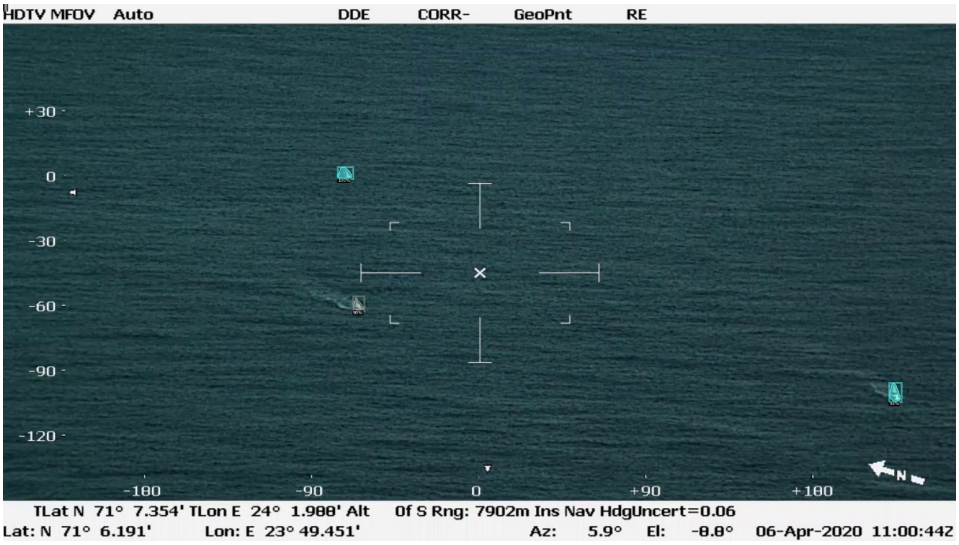

HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180 -90 0 +90 +180
N
TLat N 59° 24.648' TLon E 10° 32.437' Alt Of S Rng: 8524m Ins Nav HdgUncert=0.15
Lat: N 59° 20.437' Lon: E 10° 35.788' Az: -23.2° El: -8.1° 22-Apr-2020 13:56:542


HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180 -90 0 +90 +180
N
TLat N 59° 34.321' TLon E 10° 37.347' Alt Of S Rng: 12509m Ins Nav HdgUncert=0.19
Lat: N 59° 27.618' Lon: E 10° 36.978' Az: -1.1° El: -5.6° 22-Apr-2020 13:58:512

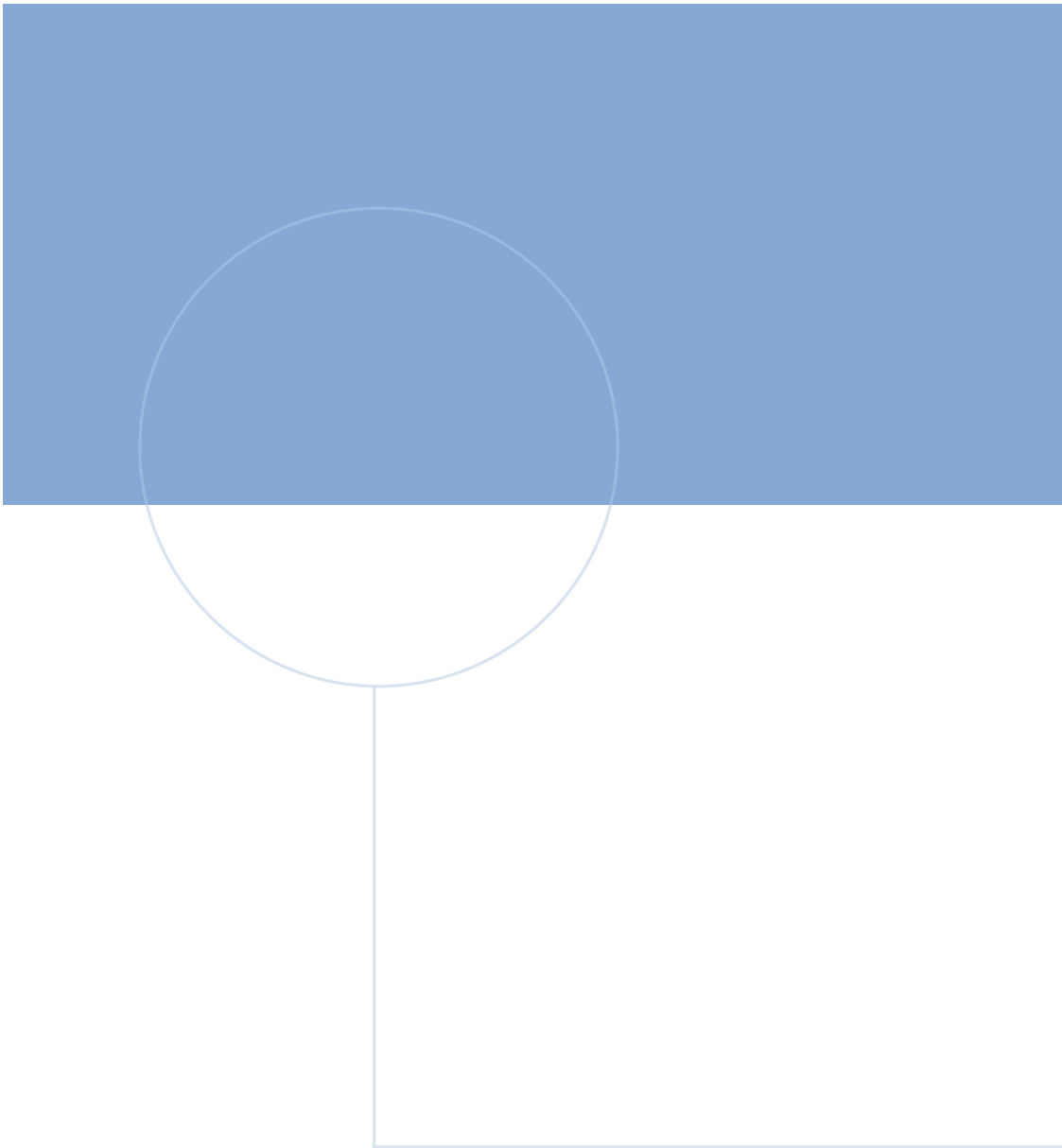

HDTV NFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 40.212' TLon E 10° 37.221' Alt Of S Rng: 13740m Ins Nav HdgUncert=0.21
Lat: N 59° 32.853' Lon: E 10° 38.061' Az: -7.0° El: -5.1° 22-Apr-2020 14:00:17Z


HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 68° 7.558' TLon E 13° 58.385' Alt Of S Rng: 4783m Ins Inertial
Lat: N 68° 7.958' Lon: E 14° 5.089' Az: 9.8° El: -13.2° 05-Apr-2020 09:55:18Z


HDTV MFOV Auto DDE CORR- Inrpt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 70° 32.178' TLon E 22° 5.076' Alt Of S Rng: 0m Ins Nav HdgUncert=0.08
Lat: N 70° 32.595' Lon: E 22° 5.541' Az: 3.4° El: -17.4° 06-Apr-2020 10:25:012


HDTV MFOV Auto DDE CORR- GeoPnt RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 71° 7.354' TLon E 24° 1.988' Alt Of S Rng: 7902m Ins Nav HdgUncert=0.06
Lat: N 71° 6.191' Lon: E 23° 49.451' Az: 5.9° El: -8.8° 06-Apr-2020 11:00:442


HDTV MFOV Auto
DDE
CORR-
GeoPnt
RE
+30
0
-30
-60
-90
-120
-180
-90
0
+90
+180
N
TLat N 59° 15.748' TLon E 10° 29.250' Alt
Of S Rng: 5971m Ins Nav HdgUncert=0.11
Lat: N 59° 14.462'
Lon: E 10° 34.860'
Az: -68.5° El: -10.9° 22-Apr-2020 13:55:18Z



NTNU

Norwegian University of
Science and Technology